Bachelorarbeit

Alexander Seidl <a_seidl@gmx.de>

Entwicklung von Webanwendungen mit Ruby on Rails am Beispiel eines GIS-Portals

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Alexander Seidl

<a_seidl@gmx.de>

Entwicklung von Webanwendungen mit Ruby on Rails am Beispiel eines GIS-Portals

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Master Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Jörg Raasch
Zweitgutachter: Prof. Dr. Michael Böhm

Abgegeben am 19. November 2007

Alexander Seidl

Thema der Bachelorarbeit
Entwicklung von Webanwendungen mit Ruby on Rails am Beispiel eines GIS-Portals
Stichworte
Ruby on Rails, Web-Framework, Web-Entwicklung, Vergleich
Kurzzusammenfassung
Die Arbeit untersucht das Web-Entwicklungsframework Ruby on Rails am Beispiel eines GIS-Portal-Prototypen. Im gleichen Umfeld auftretende Frameworks wie JavaEE und Seaside sollen vergleichend herangezogen werden. Die den Frameworks zugrundeliegenden Sprachen Ruby, Java und Smalltalk werden auf ihre angebotenen Programmiersprachen-Konzepte anhand des Prototypen verglichen. Je nach Beschaffenheit einer gegebenen IT-Umgebung eignen sich bestimmte Frameworks mehr oder weniger gut für den Einsatz. Den Abschluß der Arbeit bildet daher eine Empfehlung für die Wahl eines Frameworks.

Alexander Seidl

Title of the paper
Development of web applications with Ruby on Rails exemplified by a GIS Portal
Keywords
Ruby on Rails, Web framework, Web development, comparison
Abstract
Taking a GIS portal as example, this paper deals with the webframework Ruby on Rails. Similar frameworks like JavaEE and Seaside are examined for a comparison as well. Referring to the prototype, the associated languages Ruby, Java and Smalltalk are compared in terms of their offered programming language features. Depending on the nature of a given IT environment, a framework is more or less fitting for use. Hence, the work is finalized with a recommendation for a certain framework.

Inhaltsverzeichnis

1. Einleitung **1**
1.1. Motivation . . . 1
1.2. Abgrenzung . . . 2
1.2.1. Begriffe . . . 2
1.2.1.1. Webapplikation . . . 2
1.2.1.2. Referenzanwendung . . . 2
1.2.1.3. Sprach-Konzept . . . 2
1.2.1.4. Sprach-Konzept-Implementierung . . . 2
1.2.1.5. Anwendungs-Programmiersprache . . . 2
1.2.1.6. Web-Anwendungs-Entwicklung . . . 3
1.2.2. Methodik . . . 3
1.2.2.1. Grundannahme . . . 3
1.2.2.2. Untersuchungs-Strategie . . . 3
1.2.2.3. Kernfragen . . . 4
1.2.2.4. Erwartetes Ergebnis . . . 4
1.3. Verwandte Arbeiten . . . 4
1.4. Aufbau der Arbeit . . . 5

2. Beschreibung der Referenzanwendung **6**
2.1. Ziel der Anwendung . . . 6
2.2. Anwendungsfälle . . . 6
2.3. Design . . . 6
2.4. Umsetzung . . . 6
2.4.1. Verwendete Werkzeuge . . . 9
2.4.2. Gemachte Erfahrungen . . . 9
2.4.2.1. Arbeiten mit Ruby . . . 10
2.4.2.2. Arbeiten mit Rails . . . 10
2.4.2.3. Einbinden der Google-Maps API . . . 10
2.4.3. Fazit . . . 11

3. Programmiersprachen **12**
3.1. Die Kandidaten . . . 12
3.1.1. Ruby . . . 12

3.1.2. Java . . . 15
3.1.3. Smalltalk . . . 19
3.2. Konzepte . . . 20
3.2.1. Überblick . . . 20
3.2.2. Beschreibung und Bewertung der Konzepte . . . 22
3.2.2.1. Objektorientierung . . . 22
3.2.2.2. Uniform Access . . . 23
3.2.2.3. Generische Klassen . . . 24
3.2.2.4. Klassenvariablen- und Methoden . . . 24
3.2.2.5. Überladung von Methoden . . . 24
3.2.2.6. Überladung von Operatoren . . . 25
3.2.2.7. Typisierung . . . 25
3.2.2.8. Compiler/Interpreter . . . 27
3.2.2.9. Funktionen höherer Ordnung und Lexikalische Closures . . . 27
3.2.2.10. Continuations . . . 28
3.2.2.11. Aliasing . . . 29
3.2.2.12. Speicherbereinigung . . . 29
3.2.2.13. Reflection . . . 29
3.2.2.14. Design by contract . . . 30
3.2.2.15. Multithreading . . . 31
3.2.2.16. Reguläre Ausdrücke . . . 32
3.2.2.17. Zeigerarithmetik . . . 32
3.2.2.18. Sprachintegration . . . 33
3.2.2.19. Built-in Security . . . 33
3.2.2.20. Modul- und Klassenbibliotheken . . . 33
3.2.3. Auswahl relevanter Konzepte . . . 34

4. Sprachvergleich am Beispiel **36**
4.1. Objektorientierung - Vererbung . . . 36
4.2. Funktionen höherer Ordnung und Lexikalische Closures . . . 37
4.3. Generische Klassen . . . 39
4.4. Reflection . . . 40
4.5. Fazit . . . 41

5. Webentwicklungsframeworks **42**
5.1. Allgemein . . . 42
5.2. Besondere Paradigmen und Techniken . . . 42
5.2.1. Aspektorientierte Programmierung . . . 42
5.2.2. Inversion of Control . . . 45
5.2.3. Continuation Server . . . 46
5.3. Java Software Stack . . . 46

5.3.1. Aufbau 46
5.3.2. Fazit 48
5.4. Ruby on Rails 49
5.4.1. Philosophie 50
5.4.2. Scaffolding 50
5.4.3. Aufbau 52
5.4.3.1. Active Record 52
5.4.3.2. Action Pack 56
5.4.4. AOP in Rails 57
5.4.5. DI in Rails 58
5.4.6. Fazit 58
5.5. Seaside 59
5.5.1. Aufbau 59
5.5.2. Integration von Continuations 60
5.5.3. Die Entwicklungsumgebung 62
5.5.4. Fazit 62

6. Diskussion **64**
6.1. Ergebnisse 64
6.1.1. Sprach-Ebene 64
6.1.2. Framework-Ebene 65
6.2. Offene Fragen 65
6.3. Ausblick 66
6.4. Konklusion 66

Abbildungsverzeichnis **69**

Tabellenverzeichnis **70**

Quellcodes **71**

Glossar **73**

Abkürzungsverzeichnis **75**

Literaturverzeichnis **77**

A. Sourcecode **82**

Typografische Konventionen

- „Begriffe in Anführungszeichen“ sind feststehende Begriffe oder zusammengehörende Wortgruppen.
- Schrift mit fester Breite weist auf Code hin, der genauso geschrieben werden muß: `public static void main(String args[]){...}`
- Abkürzungen wie GIS können im Abkürzungsverzeichnis nachgeschlagen werden.
- Bei unklaren Begriffen lohnt sich ein Blick ins Glossar.

1. Einleitung

1.1. Motivation

Seit der Entwicklung des Internets bis heute im Jahre 2007 besteht der Wunsch in der Geschäftswelt, Produkte und Dienstleistungen virtuell und damit global zu präsentieren. Zu Beginn geschah dies einfach durch Lagern von statischen HTML-Dateien auf einem Webserver und der Möglichkeit die dort gespeicherten Informationen weltweit mithilfe eines Browsers aufzurufen. Im Laufe der Zeit stiegen jedoch die Anforderungen. Produkte sollten nun nicht mehr einfach präsentiert, sondern auch zum Verkauf angeboten werden können. Komplexe Bestellvorgänge mußten abgewickelt werden. Dies beinhaltete unter anderem die Speicherung der Bestellungen und der persönlichen Daten der Kunden. Über das CGI-Protokoll konnten Webserver Eingabedaten an ein externes Programm übergeben, welches diese verarbeite und eine passende HTML-Antwort generierte. Kundendaten, Bestellungen und passende Bestätigungen über den Bestellvorgang konnten auf diesem Wege gespeichert und erzeugt werden. Schon bald reichte auch dies nicht mehr aus. Es stellte sich heraus, daß die CGI-Programme immer die gleichen Probleme lösen und daher um bestimmte Aspekte erweitert werden mußten. Unter anderem: Prüfung der Eingabedaten auf Korrektheit und bösartigen Code (Sicherheit). Trennung der HTML-Ausgabe vom verarbeitenden Code, um Arbeitsteilung zwischen Programmierern und Designern zu ermöglichen (MVC-Paradigma). Zwischenspeicherung von häufig angefragten Informationen (Caching). Darstellung der Informationen in verschiedenen Sprachen (Internationalisierung und Lokalisierung). Eine neue Art von Anwendung wurde geboren: Die Webapplikation. Um solche Anwendungen effizienter zu entwickeln, wurden passende Webentwicklungs-Frameworks geschaffen, die die Lösung der immer wiederkehrenden Aufgaben vereinfachen sollten. Die Programmiersprache Java und auf ihrer Grundlage entwickelte Framworks wie Hibernate (Persistenz) oder Struts (MVC u.a.) sicherten sich als erstes die Marktführerposition. Die Ansprüche an Webapplikationen wuchsen weiter. Die Behandlung von Web-Services, verteilter Transaktionen und moderner Paradigmen wie Aspektorientierte Programmierung sollte nun auch integriert und vereinfacht werden. Die Java-Welt reagiert mit dem Hinterherschieben von immer weiteren Frameworks, die alle erdenklichen Anforderungen abdecken. Das Ergebnis: Produktivitätseinbußen infolge gesteigerter Komplexität. Scheinbar als eine Gegen-Antwort darauf drängte das Webentwicklungs-Framework „Ruby on Rails“ auf den Markt.

1.2. Abgrenzung

1.2.1. Begriffe

1.2.1.1. Webapplikation

Im Rahmen der Arbeit wird oft der Begriff „Web-Applikation“[1] verwendet werden. Hiermit ist im wesentlichen eine serverseitige Anwendung gemeint, die über einen Webserver Client-Anfragen mit Informationen aus einer Datenbank oder einer anderen Informationsquelle bedient.

1.2.1.2. Referenzanwendung

Ein Prototyp einer „Ruby on Rails“-Anwendung soll als Untersuchungs-Grundlage herangezogen werden. Es handelt sich um eine Geo-Informations-System für Trekking-Aktivitäten (GIS-TA). Er dient in dieser Arbeit lediglich explorativen Zwecken und ist nicht für die Produktion geeignet. Seine Beschreibung in Abschnitt 2 ist daher als eine Vision zu sehen.

1.2.1.3. Sprach-Konzept

Mit Sprach-Konzept ist eine Eigenschaft von Programmiersprachen gemeint. Eigenschaften sind zum Beispiel die Art der Typisierung einer Sprache, die Paradigmen wie Objektorientierung, die sie unterstützt oder das Bereitstellen von Closures.

1.2.1.4. Sprach-Konzept-Implementierung

Sprach-Konzepte sind in Sprachen unterschiedlich implementiert. Zum Beispiel werden Closures [Wikipedia (2007d)] in Ruby und Smalltalk durch „Blöcke“, in Java durch „Anonyme Klassen“ implementiert.

1.2.1.5. Anwendungs-Programmiersprache

Anwendungs-Programmiersprachen sind Sprachen, in denen man sich im Gegensatz zu den hardwarenahen oder den System-Programmiersprachen weitgehend auf fachliche Aspekte eines zu lösenden Problems konzetrieren kann. Der Zugriff auf Ressourcen erfolgt über Namen.

[1] Synonym: „Web-Anwendung“

Eine Beschäftigung mit der Art der Aufbewahrung der Ressourcen (Festplatte, Datenbank, Netzwerk, ...) ist fast nicht mehr notwendig, da über DSLs von Implementierungs-Details abstrahiert wird.

1.2.1.6. Web-Anwendungs-Entwicklung

Entwicklung, die sich um die speziellen Belange der Web-Programmierung dreht. Dazu gehören u.a. die Verwaltung des Zustandes von HTTP-Sessions, das Lesen von HTTP-Anfragen, das Senden von HTTP-Antworten, das Lesen und Schreiben von Daten in eine Datenbank. „Webprogrammierung“ wird synonym verwendet.

1.2.2. Methodik

1.2.2.1. Grundannahme

Dieser Arbeit liegt die Annahme zugrunde, daß sich eine elegante Implementierung programmiersprachlicher Konzepte förderlich auf die Produktivität, Verständlichkeit und Erlernbarkeit einer Sprache auswirkt. Mit „elegant“ ist vor allem gemeint, daß sich die Implementierung mit wenig Code umsetzen läßt. Auf Eleganz wird daher bei der Untersuchung der Konzepte in dieser Arbeit geachtet. Elegant wird dann gleichgesetzt mit produktiv, verständlich und leicht erlernbar. Eine elegante Sprache ist daher eine Sprache, die gut als Web-Entwicklungssprache eingesetzt werden kann.

1.2.2.2. Untersuchungs-Strategie

Es soll daher folgender Ansatz verfolgt werden: In einem ersten Schritt werden explorativ Eigenschaften der zugrundeliegenden Sprachen Java, Ruby und Smalltalk untersucht. Dann werden wichtige Sprachkonzepte beschrieben. Für jedes der beschriebenen Konzepte soll bewertet werden, ob die jerweiligen Implementierungen der Sprachen auf Eleganz hin verglichen werden können und ob sie relevant für die Domäne der Web-Anwendungs-Entwicklung sind. Aufgrund der Bewertungen findet im Anschluß eine Auswahl von vier Konzepten statt, anhand derer unter Rückbezug auf die Referenzanwendung die Sprachen auf ihre Eleganz hin untersucht werden. Erste Stärken und Schwächen für die Entwicklung von Webapplikationen im Rahmen eines Frameworks sollen so ersichtlich werden. Daraufhin werden die auf den Sprachen aufbauenden Frameworks, also ein Java-Software-Stack, Ruby on Rails und Seaside beschrieben und untersucht. Auch hier soll geklärt werden, wo die Unterschiede liegen und welche Konzepte angeboten werden. Die parallele Untersuchung von Smalltalk und Java und

ihrer Frameworks hat den Zweck, die Besonderheiten von Ruby und seinem Framework „Ruby on Rails“ besser hervortreten zu lassen. Dies führt uns zu den Kernfragen der Arbeit im nächsten Abschnitt.

1.2.2.3. Kernfragen

1. Gibt es zwischen Ruby, Smalltalk und Java Unterschiede hinsichtlich der Implementierung von Programmiersprachen-Konzepten? Wie wirkt sich die Art und die Konsequenz der Umsetzung auf ihre Eignung als Web-Anwendungs-Programmiersprache aus? Ist sie produktiv einsetzbar oder wirkt sich die konkrete Umsetzung einiger Konzepte hinderlich aus?
2. Wie ist Ruby on Rails aufgebaut und welche Merkmale weist es auf? Welchen Aufbau und welche Besonderheiten haben Frameworks anderer Sprachen wie Java (JavaEE) und Smalltalk (Seaside). Welche Unterschiede gibt es besonders in Hinblick auf die erreichbare Produktivität und den Leistungsumfang.
3. Für welche IT-Umgebung eignet sich welches Framework?
4. Wie läßt sich ein GIS in Form der Google-API in Rails integrieren?

1.2.2.4. Erwartetes Ergebnis

Am Ende der Betrachtungen soll Klarheit über die Sprachkonzepte von Ruby, Java und Smalltalk bestehen und wie die Implementierungen in den jeweiligen Sprachen die Framework-Programmierung begünstigen oder behindern. Außerdem müssen Unterschiede, Stärken und Schwächen der dazugehörigen Frameworks hervortreten. Verantwortliche sollten dann in der Lage sein dasjenige Framework auszusuchen, welches den Anforderungen ihres Arbeitsumfeldes am besten gerecht wird.

1.3. Verwandte Arbeiten

Bruce Tate beschäftigt sich in seinem Buch „Beyond Java“ [Tate (2005)] mit einschlägigen Fragen. Ein Großteil der Arbeit gründet auf seinen Ergebnissen.

1.4. Aufbau der Arbeit

Kapitel 2 „Beschreibung der Referenzanwendung“: Das Kapitel beschreibt den geplanten Aufbau der Referenzanwendung.

Kapitel 3 „Programmiersprachen“: Hier soll ein Überblick über die untersuchten Sprachen gegeben werden. Erste Unterschiede und Besonderheiten sollen hier zutage treten. Desweiteren findet hier eine detaillierte Betrachtung von Programmiersprachenkonzepten statt. Die Konzepte werden dabei auch auf ihre Eignung als Untersuchungskonzept hin untersucht und bewertet. Am Ende des Kapitels werden vier Konzepte für einen Vergleich ausgewählt.

Kapitel 4 „Sprachvergleich am Beispiel“: In diesem Kapitel werden die Programmiersprachen anhand der in Kapitel 3 ausgewählten Konzepte verglichen.

Kapitel 5 „Webentwicklungsframeworks“: Ein Java-Software-Stack, Ruby on Rails und Seaside werden beschrieben und bewertet. Besonderheiten werden genannt.

Kapitel 6 „Diskussion“: Die Ergebnisse werden zusammengefaßt. Ein Blick in zukünftige Entwicklungen wird versucht und eine Empfehlung abgegeben.

2. Beschreibung der Referenzanwendung

2.1. Ziel der Anwendung

Ziel der Anwendung ist es an einem nichttrivialen Beispiel anhand von im folgenden noch zu erarbeitenden Sprach-Konzepten die Potenz von Ruby on Rails zu zeigen. Es handelt sich um eine Client-Server-Anwendung, welche ein Geo-Informations-System für Trekking-Aktivitäten, im folgenden GISTA genannt, darstellt. Es ist geplant, daß sich Internet-Nutzer registrieren können, um einen Account zu erhalten. Anschliessend können sie auf einer durch die Google-Maps-API zur Verfügung gestellten Karte Punkte und Routen markieren, benennen, beschreiben und anderen Nutzern zeigen. Auf diese Weise soll eine Gemeinschaft aufgebaut werden, die eine gemeinsame Leidenschaft, das Bereisen neuer Orte, teilen.

2.2. Anwendungsfälle

Tabelle 2.1 listet alle geplanten Anwendungsfälle des GISTA auf. Es handelt sich um acht Anwendungsfälle, die zusammen eine sinnvollen Dienst anbieten. Es sei hier noch einmal darauf hingewiesen, daß die Anwendung lediglich den Zweck hat die Eigenschaften der Sprache Ruby zu untersuchen und bei ihr daher kein Schwerpunkt auf Ausgereiftheit gelegt wird.

2.3. Design

Der fachliche Aufbau des GISTA kann Abbildung 2.1 entnommen werden. Die Bedienoberfläche ist in diesem Fall der Internet-Browser. Das Rails-Framework, auf das später noch genauer eingegangen wird, leitet die Anfragen von der Oberfläche zu den fachlichen Komponenten (siehe Tabelle 2.2) und stellt die Verbindung zum Datenspeicher her.

2.4. Umsetzung

#	Anwendungsfall
	Registrierung in Login
1	Nutzer registriert sich am System mit einem beliebigen Nutzernamen und einem Passwort. Per Email erhält er einen Link, um seine Registrierung zu bestätigen.
2	Über ein Formular kann sich der Nutzer einloggen.
	Nutzung
3	Eine durch die Google-Maps API bereitgestellte Karte (im folgenden einfach nur „Karte“) kann der Nutzer per Adresseingabe an einem beliebigen Punkt der Erde zentrieren.
4	Der Nutzer wählt per Linksklick der Maus auf der Karte einen Punkt, im folgenden „Ort“ genannt. Diesem wird über ein Formular ein Titel und eine Beschreibung zugewiesen. Der Ort wird auf der Karte mit einem Marker sichtbar ausgezeichnet. Der Nutzer entscheidet nun, wie er mit dem Ort weiterverfährt: Editieren, Speichern, Löschen stehen zur Verfügung. Nach dem Speichern wird sein Ort mit Beschreibung und dem Titel serverseitig gespeichert.
5	Über eine auf der Oberfläche sichtbaren Liste, kann der Nutzer seine gespeicherten Orte einsehen und sie auf Wunsch auf der Karte ein- oder ausblenden.
6	Statt nur einen einzigen kann der Nutzer auch mehrere Orte setzen, die dann eine Route bilden. Jeder Ort wird wie gehabt mit Basisinformationen hinterlegt. Desweiteren wird auch die Route selbst mit Titel und Beschreibung versehen. Wie bei einem einzelnen Ort sind auch hier ein anschließendes Editieren, Speichern und Löschen möglich.
7	Für jeden Ort und jede Route, kann der Nutzer entscheiden, ob er diesen mit Titel und Beschreibung für die Community sichtbar macht.
8	Über ein Suchfeld kann jeder Nutzer nach anderen Nutzernamen suchen und so deren öffentlich gemachten Orte und Routen in einer Liste sichtbar machen.

Tabelle 2.1.: Anwendungsfälle des GISTA

Komponente	Beschreibung
Registrierung	Verwaltet den Registrierungsprozeß eines neuen Nutzers
Login	Verwaltet den Authentifikationsprozeß eines Nutzers
Kartenmodul	Das Kartenmodul ermöglicht die Speicherung von Orten und Routen, ermöglich eine geeignete Zentrierung der Karte und das Suchen nach Orten und Routen anderer Nutzer

Tabelle 2.2.: Komponenten des GISTA

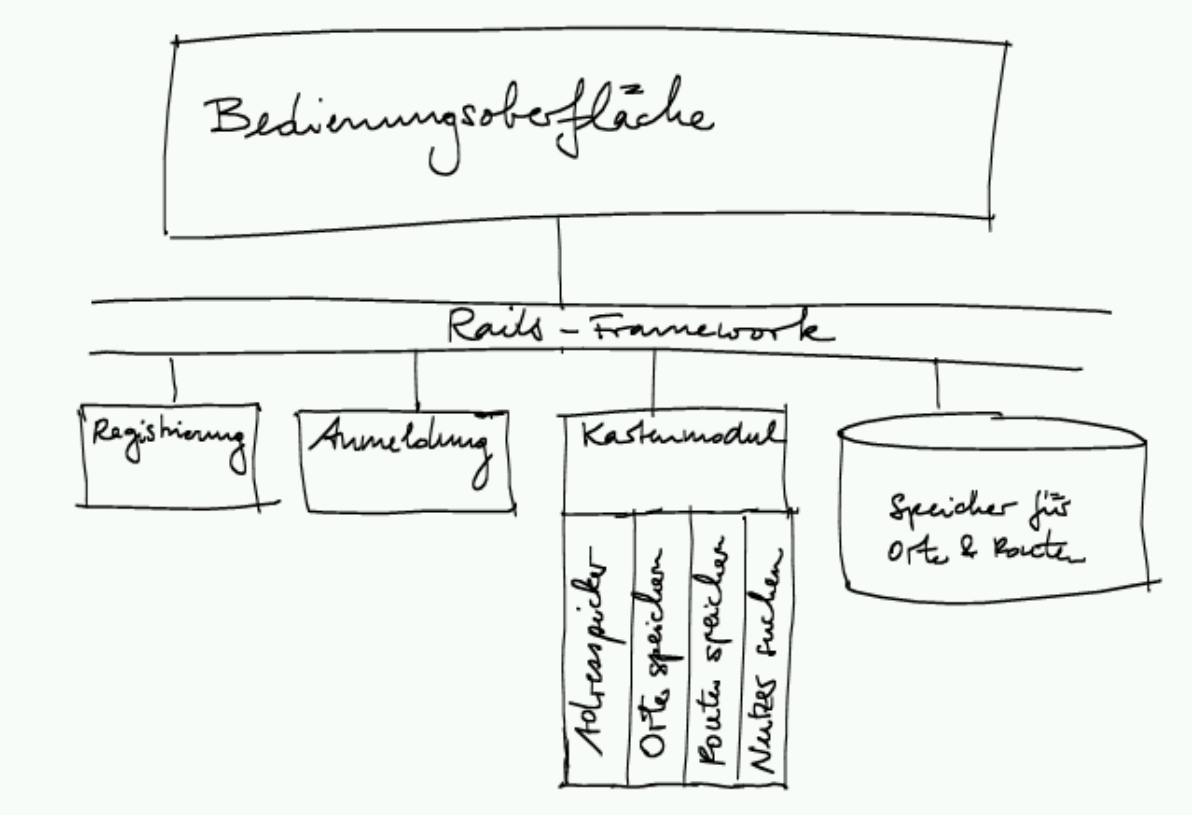

Abbildung 2.1.: GISTA: Fachliche Architektur

2.4.1. Verwendete Werkzeuge

Für die Entwicklung des GISTA wurden die in Tabelle 2.3 aufgeführten Werkzeuge verwendet.

Werkzeug	Beschreibung	Version	Quelle
Windows XP	Windows XP wurde als Ausführungsplattform des GISTA genommen	XP SP2	
Rubyinterpreter	Es wurde ein selbstextrahierender Rubyinterpreter für Windows gewählt	1.8.6	[Opensource (b)]
Railsframework	Das MVC-Web-Framework, auf dem die Entwicklung des GISTA basiert	1.2.3	Gem-Paketmanager mittels `gem install rails -v=1.2.3`
Google-Maps-API	Die von Google öffentlich gemachte Programmierschnittstelle, um Geocoding zu betreiben	aktuelle Fassung	[Google (b)]
Eclipse IDE	Open-Source-Framework zur Entwicklung von Software nahezu aller Art	3.2.0	[Foundation]
Eclipse Plugin Radrails	eine integrierte Entwicklungsumgebung für das Ruby in Rails Framework	0.7.2	[Aptana]
Eclipse Plugin RDT	Ruby Development Tools für Eclipse.	0.9.0	[Opensource (c)]
Mysql-Server	Relationale Datenbank	5	[AB]

Tabelle 2.3.: Verwendete Entwicklungsumgebung

2.4.2. Gemachte Erfahrungen

Erste vom Autor der Arbeit gemachte Eindrücke bei der Entwicklung des Prototypen beziehen sich auf sprachliche Aspekte der Programmiersprache Ruby, die gebotenen Leistungen des Web-Frameworks Ruby on Rails sowie die Benutzung und Einbindung der Google-Maps-API in das Rails-Framework. Diese sollen hier kurz beschrieben werden.

2.4.2.1. Arbeiten mit Ruby

Ruby besitzt eine einfache und klare Syntax. Nach Einarbeitung in die elemtaren Konzepte der Sprache ging das Programmieren leicht von der Hand, da sich Ruby oft so verhielt, wie man es intuitiv vermutet hatte. Bevor ein Blick in die API notwendig war, konnte man sich auch der „Interaktiven Ruby-Shell“ (irb-Shell), einem Kommandozeileninterpreter für Ruby, bedienen. Analysieren und Testen von Ruby-Code-Schnipseln war mit ihm leicht möglich.

2.4.2.2. Arbeiten mit Rails

Das Arbeiten mit dem Web-Framework Rails wird unter anderem durch die Verwendung von Scaffolding-Scripts[2] sehr komfortabel. Die grundlegende Struktur eines „Ruby on Rails“-Projektes kann beispielsweise einfach mittels diverser Konsolenbefehle erfolgen. `rails gista` legt im aktuellen Verzeichnis alle benötigten Projekt-Verzeichnisse an. Durch `ruby script/generate migration create_user` kann eine Datenbank-Migration erstellt werden. In ihr wird die Definition einer Datenbank-Tabelle mittels Ruby-Code festgelegt. Gleichzeitig wird ein Muster für das User-Model in der Verzeichnisstruktur erstellt. `rake db:migrate` überträgt diese Defintion auf das Datenbankschema. `ruby script/generate scaffold User` erledigt den Rest und erstellt Dateien für die View- und die Controller-Schicht und ordnet sie in die passenden Verzeichnisse. Der UsersController ist auch schon mit passenden Methoden für elemantare CRUD-Operationen[3] ausgestattet. Nach diesen wenigen Handgriffen kann man im Browser auch schon die URL `http://localhost/users/new` eingeben und erhält ein simples Formular, in dem für jedes Model-Attribut Eingaben vorgenommen werden können, die man anschließend speichern kann. Das durch den Scaffold-Mechanismus generierte Codegerüst ist eine gute Grundlage für weitere Anpassungen.

2.4.2.3. Einbinden der Google-Maps API

Google-Maps basierte Anwendungen beziehen Javascript-Bibliotheken von Google[4]. In diesen befindet sich die Logik, mit deren Hilfe vielfältige Aufgaben [Lewis u. a. (2007)] im Rahmen des Geocodings vorgenommen werden können. Die Einbindung des Google-Maps-Codes ist mit wenigen Handgriffen erledigt. Im Kopf der HTML-Seite wird ein `script`-Tag mit einem `src`-Attribut definiert. Dieses verweist auf eine URL, von der der Browser die Javascript-Bibliotheken für das Arbeiten mit Google-Maps bezieht. Desweiteren muss ein Schlüssel, der

[2] Scaffold: engl. für Gerüst
[3] CRUD Akronym für „Create, Read, Update, Destroy“
[4] genaugenommen von maps.google.com

sogenannte „Google Maps Key“ angegeben werden, den man sich hier [Google (a)] besorgen kann. Er ist notwendig, da er eine bestimmte Internet-Domain für die Benutzung der API freischaltet. In dem von Rails generierten Verzeichnis `/public/javascripts` im Anwendungsverzeichnis werden serverseitige Javascript-Funktionen abgelegt. In der dortigen Datei `application.js` liegen die Funktionen zum Setzen von Markern auf die Karte oder zum Zeichnen von Linien oder Flächen.

2.4.3. Fazit

Das Arbeiten mit dem Rails-Framework fühlt sich an wie das Konfigurieren einer Anwendung. Der meiste Code ist schon geschrieben. Nur an den richtigen Stellen im Verzeichnisbaum müssen einige Anpassungen vorgenommen werden, um das gewünschte Verhalten zu erzielen. Unzählige Scripte und Funktionen aus dem Railsframework unterstützen einen bei der automatischen Generierung von Code für Controller, für HTML-Code im View-Layer, für das Anlegen von Models und Datenbankmigrationen und vielen weiteren domänenspezifischen Aufgaben wie Eingabevalidierung und Cookiemanagment. Die zugrundeliegende Programmiersprache Ruby glänzt durch Eingängigkeit und leichte Erlernbarkeit. Die Einbindung einer Google-Maps-Funktionalität beschränkt sich auf das Laden einiger Javascript-Dateien.

Die hier gemachten Aussagen sind das Ergebnis erster gemachter Eindrücke des Autors. Eine systematische, vergleichende Untersuchung der Sprache Ruby und des Railsframeworks mit äquivalenten Lösungen in Java und Smalltalk soll nun die Frage klären, wo die Unterschiede liegen. Im folgenden Abschnitt werden zu diesem Zweck die Sprachen Ruby, Smalltalk und Java verglichen.

3. Programmiersprachen

Das folgende Kapitel 3.1 soll ein kurzen stichpunktartigen Überblick über Ruby, Java und Smalltalk liefern. Dabei wird teilweise anhand von Beispielen, teilweise anhand von weiteren Erläuterungen kurz auf besondere Konzepte und Merkmale der Sprachen hingewiesen und eventuell Probleme und Unterschiede zu den anderen Sprachen genannt. Diese gedankliche Vorarbeit soll ein erstes Gefühl für die Sprachen vermitteln und die Grundlage für zwei Schritte im Anschluß an diesen Abriß legen. Zum einen sollen die Konzepte dann zusätzlich definiert und abgegrenzt werden, um anschließende Untersuchungen auf eine einheitliche Definitionsbasis zu stellen. Zum anderen sollen durch diesen explorativen Ansatz Stärken und Schwächen der Sprachen zutage gefördert werden, um später eine Auswahl von signifikanten Konzepten zu erleichtern, anhand derer die drei Sprachen unter Rückbezug auf die Referenzanwendung exemplarisch verglichen werden.

3.1. Die Kandidaten

3.1.1. Ruby

Es folgen besondere Merkmale der Programmiersprache Ruby. Zur Verdeutlichung werden einige Merkmale mithilfe der „Interactive Ruby Shell“ [Opensource (a)], auch „irb“ genannt, exemplifiziert.

1. vollständig objektorientiert
2. Skriptsprache
3. stark, dynamisch typisiert
4. Bereiche[5]: Sind in Ruby Objekte.
5. Reguläre Ausdrücke: Sind in Ruby Objekte.
6. Datenstrukturen: Arrays, Hashes. In ihnen brauchen Objekte nicht homogen zu sein.

[5] engl. Ranges

7. Dateien[6]: Durch sie zu iterieren ist wie durch „Collections“ zu iterieren.
8. Klassen können zur Laufzeit erweitert werden.
9. Mixins
10. Aliasing[7]
11. Metaprogrammierung

zu 1: Alles ist in Ruby ein Objekt, dem eine Nachricht geschickt werden kann.

```
irb(main):001:0> 4.444.round
=> 4
irb(main):013:0> nil.class
=> NilClass
```

Quellcode 3.1: In Ruby ist alles ein Objekt

zu 2

Führt zu kurzem „Code Compile Cycle“.

zu 3: Ruby ist stark und dynamisch typisiert. Die Zuweisung `n=1` erfolgt ohne Angabe eines Typs (dynamisch). Die Addition `n+4` wird von Ruby abgelehnt, da hierfür ein interner Typecast stattfinden müsste, was einer schwachen Typisierung entspräche.

```
irb(main):014:0> n=1
=> 1
irb(main):015:0> n.class
=> Fixnum
irb(main):016:0> n="fish"
=> "fish"
irb(main):017:0> n.class
=> String
irb(main):018:0> n+4
n+4
TypeError: can't convert Fixnum into String
 from (irb):18:in `+'
 from (irb):18
```

[6] engl. Files
[7] engl. „Method interception“

```
irb(main):019:0>
```

Quellcode 3.2: Ruby ist stark, dynamisch typisiert

zu 4: Bereiche sind in Ruby Objekte. Mithilfe des === Operators kann geprüft werden, ob ein Objekt Element des Bereiches ist.

```
irb(main):019:0> range=1..3
=> 1..3
irb(main):020:0> range.class
=> Range
irb(main):021:0> range === 1
=> true
irb(main):022:0> range === 4
=> false
```

Quellcode 3.3: Ruby Bereiche

In Java kann man nicht auf Bereiche als Objekte zurückgreifen.

```
class Range {
 public static void main(String[] args) {
  int i = 4;
  if (1 <= i && i <= 3){
   System.out.println("true");
  }
  else
  {
   System.out.println("false");
  }
 }
}
```

Quellcode 3.4: Java hat keine Bereiche

zu 7: Das Schließen der Datei ist nicht notwendig. Genausowenig wie das Arbeiten mit Ausnahmen für den Fall, daß kein Dateiname übergeben wurde oder diese nicht vorhanden ist. Ruby erledigt diese Sachen im Hintergrund, während man in Java darauf Rücksicht nehmen muss. Aufruf: `ruby grep.rb filename regex`.

```
File.open(ARGV[0]) do |file|
```

```
    rx = Regexp.new(ARGV[1])
    while line=file.gets
      puts line if line =~ rx
    end
  end
```

Quellcode 3.5: Ruby Files

zu 9: Mixins sind Module, die zusammengehörige Methoden logisch gruppieren und so einen Aspekt separieren. Sie können von anderen Klassen inkludiert werden als eine Art Hineinmischung. Man könnte sie auch als „Interfaces“ mit Implemetierung bezeichnen. Sie können nicht instaziert werden und haben ihren eigenen Namensraum. Mit ihnen kann Mehrfachvererbung simuliert werden.

zu 10: In dem folgenden Rubycode-Beispiel 3.6 wird die Konstruktor-Methode `new` nach `original_new` umbenannt. In der anschließend definierten neuen Methode `new` kann zusätzliches Verhalten implementiert werden. Durch dieses Konzept ist es möglich das Verhalten einer Klasse zu ändern ohne deren Code zu ändern.

```
  class Class
    alias_method :original_new, :new
    def new(*args)
      result = original_new(*args)
      print "Unattended␣laptop␣error."
      return result
    end
  end
```

Quellcode 3.6: Ruby Aliasing

Listing 3.6 erzeugt folgende Ausgabe:

```
irb(main):015:0> i=[1,2,3]
i=[1,2,3]
Unattended laptop error.Unattended laptop error.Unattended...
```

3.1.2. Java

Es folgen besondere Merkmale der Programmiersprache Java.

1. hybrid objektorientiert

2. komplilierte Sprache.
3. Java Virtuelle Maschine: „compile once, run anywhere“
4. stark, statisch typisiert
5. aspektorietierte Programmierung (AOP)
6. Reflection
7. Unterscheidung zwischen „Schnittstellen“ und „Klassen“
8. umfangreiche Klassenbibliothek

zu 1: Die Grunddatentypen (int, boolean usw.) sind keine Objekte.

zu 2: Das führt zu einem längeren „Code Compile Cycle“.

zu 3: Die JVM ist essentieller Bestandteil der Java-Laufzeitumgebung. Java-Code kompiliert zu Bytecode, der von einer an die Plattform angepaßten JVM interpretiert wird. Daher laufen Java-Programme in der Regel ohne weitere Anpassungen auf verschiedenen Computern und Betriebssystemen, für die eine JVM existiert (Plattformunabhängigkeit). Der andere Teil der Java-Laufzeitumgebung sind die Java-Klassenbibliotheken.

Das Konzept der JVM-Architektur weist einige signifikante Vorteile auf.

- Sicherheit Pufferüberlauf: Suns JVM überwacht zur Laufzeit die Ausführung des Programms und verhindert Pufferüberläufe, also daß ein Programm über Arraygrenzen hinweg liest oder schreibt, was von Angreifern dazu benutzt werden kann fremden Code auf einem System auszuführen.
- Sicherheit bei Ausführung remoten Codes: Die JVM erlaubt eine sehr feingranulare Kontrolle über die Aktionen, die Code innerhalb der Host-Maschine ausführen darf. Dadurch kann Code aus unsicheren Quellen wie in einem „Sandkasten“ ausgeführt werden ohne Schaden azurichten.
- Klassenbibliotheken: Sprachen, die sich an Java anlehnen und Bytecode generieren, können auf der JVM ausgeführt werden und von dem Reichtum der Java-Klassenbibliotheken profitieren.
- „Garbage Collection“: Die automatische Speicherbereinigung ist Bestandteil der JVM

zu 4: Genau wie Ruby ist Java stark typisiert, fordert also typkompatible Behandlung aller Objekte, um so Typsicherheit zu garantieren.[8] Durch die statische Typisierung müssen alle Variablen, Funktionsparamter und Rückgabewerte bereits zur Kompilierzeit mit einem Typ versehen werden.

[8] was durch implizite „Typecasts“ jedoch umgangen werden kann

Wichtige Probleme: Bis Version 5.0 konnte Java zur Kompilierzeit keine Typsicherheit für Objekte in Collections garantieren (Listing 3.7). Sobald das String-Objekt `elephant` der Collection hinzugefügt wurde, verlor es seine Typinformation, woraus typinkompatible Behandlung des Objektes resultieren konnte, nachdem man das Objekt wieder aus der Collection herausholte. Der Programmierer war daher beim Herausholen des Objektes durch einen Typecast selbst dafür verantwortlich das Objekt korrekt zu retypisieren.

```
ArrayList animals = new ArrayList();
animals.add("elephant");
String cat = (String)animals.get(0);
```

Quellcode 3.7: Java: Typsicherheit in Collections vor Java 5.0

Seit Java Version 5.0 gibt es das Konzept der „Generics“, dessen Anwendung aus Codebeispiel 3.7 das folgende werden läßt:

```
ArrayList<String> animals = new ArrayList<String>();
animals.add("elephant");
String elephant = animals.get(0);
```

Quellcode 3.8: Java: Typsicherheit in Collections nach Java 5.0 mit Generics

Durch ein Konzept namens „Type Erasure“ kommt es jedoch bei der Benutzung von Generics intern trotzdem zu einem Verlust der Typinformation. Die ArrayList in Codebeispiel 3.8 bleibt daher eine Collection aus „Object“-Objekten und keine Collection aus „String“-Objekten.

Resultierende Probleme:

- Die Typsicherheit geht in folgender Code-Zeile nicht von genericList nach notGeneric über, obwohl sie beide intern auf dieselbe Liste im Speicher zeigen: `List notGeneric = genericList;`
- `ArrayList<int> al = ArrayList<int>();` geht nicht, da Primitive keine generischen Datentypen parametrisieren können.
- Beide Listen gehören in folgender Code-Zeile zum selben Typ ArrayList: `ArrayList<String>` und `ArrayList<Book>`
- Die JVM hat wegen TE keine Kenntnis von den Paramtern generischer Typen, sodaß andere Klassen via Reflection keinen Zugriff auf die Typinformationen haben.

Generics sind inkonsequent umgesetzt, da sie nicht auf Bytecode-Ebene durchgezogen werden. Sie führen lediglich zu einer zusätzlichen syntaktischen Bürde. Eine genaue Analyse der

Java Generics erfolgt in [Subramaniam (2005)]. Der Autor dieses Artikels, Venkat Subramaniam, schreibt:

> „Generics in Java were created to provide type-safety. They are implemented only at the language level. The concept is not carried down to the byte code level. It was designed to provide compatibility with legacy code. As a result, generics lack what they were intended for, type-safety."

zu 5: AOP [Schempp (2006)] ist ein Konzept, welches einem POJO beliebiges Verhalten hinzufügen kann ohne dessen Code zu verändern. Dadurch können objektweit querschneidene Belange, auch Aspekte genannt, implementiert werden. Solche Aspekte betreffen z.B. das Logging, das Transaktionsmanagment und die Sicherheit. Diese Feature erkauft man sich jedoch durch die Hinzunahme des Java-Frameworks „Spring", welches zusätzlichen Einarbeitungsaufwand erfordert. In Ruby kann AOP-Verhalten durch einfache Konzepte wie „Aliasing" und „Mixins" realisiert werden.

zu 6: Java bietet eine Reflection-API als Bestandteil der Laufzeitumgebung. Damit ist es möglich, zur Laufzeit auf Klassen und Methoden zuzugreifen, deren Existenz oder genaue Ausprägung zur Zeit der Programmerstellung nicht bekannt war. Listing 3.9 zeigt eine Methode, die eine beliebige andere Methode eines gegebenen Objekts aufruft und deren Rückgabewert zurückgibt.

```
public String getStringProperty(Object object, String methodname)
{
   String value = null;
   try
   {
     Method getter = object.getClass().getMethod(methodname, new Class[0]);
     value = (String) getter.invoke(object, new Object[0]);
   } catch (Exception e) {}
   //Fehlerbehandlung zwecks Übersichtlichkeit nicht implementiert.

   return value;
}
```

Quellcode 3.9: Java Reflection

Aufruf:

```
System.out.println("Vorname von " +
person +
" ist " +
```

```
getStringProperty(person, "getVorname")
);
```

Der Aufruf durch die obige Zeile würde dann die Methode getVorname() des Objekts person aufrufen und deren Rückgabewert ausgeben.

In Ruby implemetiert man es so:

```
print "Vorname von " +
   person +
   " ist " +
   person.send(:getVorname) if person.respond_to?(:getVorname)
```

Quellcode 3.10: Ruby Reflection

Hier deuten sich Stärken von Ruby im Bereich Reflection und Metaprogrammierung an.

zu 7: Schnittstellen und Klassen werden in Java explizit unterschieden. Eine Klasse kann beliebig viele Schnittstellen implementieren, hat aber stets genau eine Basisklasse. Java unterstützt keine Mehrfachvererbung, da lediglich Schnittstellen, jedoch keine Klassen mehrfach vererbt werden können.

zu 8: Durch die Klassenbibliothek wird dem Programmierer eine einheitliche, vom zugrunde liegenden Betriebssystem unabhängige Schnittstelle (API) angeboten. Die Java-Klassenbibliothek ist umfangreich und bietet Unterstützung für eine Vielzahl von Problemen.

3.1.3. Smalltalk

Es folgen besondere Merkmale der Programmiersprache Smalltalk.

1. rein objektorientiert
2. stark, dynamisch typisiert
3. VM bytecodeinterpretiert
4. Smalltalk kommt zusammen mit einer vollständigen IDE.
5. Kontrollstrukturen werden durch Blöcke realisiert.

zu 1: In Smalltalk ist alles ein Objekt: Zeichenketten, Integer, Boolesche Werte, Klassen, ausführbarer Code, Stack-frames, der Speicher, Codeblöcke etc. In Ruby ist das ebenso. Objekte werden dadurch aktiviert, daß man ihnen Nachrichten (Messages) schickt. Dies führt dazu, daß

die entsprechende Methode des Objekts ausgeführt wird. Nur sehr wenige Methoden greifen auf echte Primitive der VM zurück.

zu 3: Smalltalkprogramme werden in Bytecode übersetzt, der durch eine virtuelle Maschine ausgeführt wird. Dadurch laufen Smalltalk-Programme ohne jegliche Änderung auf jedem System, für das eine virtuelle Maschine existiert. Ursprünglich wurde der Bytecode interpretiert. Kommerzielle Umgebungen arbeiten inzwischen fast ausschließlich mit dynamischer Übersetzung.

zu 4: Smalltalk-Objekte können zur Laufzeit verändert werden. Das bedeutet man kann eine aufgetretene Exception durch Code-Änderung debuggen, einen „Go“-Knopf in der IDE drücken und der Prozess fährt an der Stelle fort, wo das Problem ursprünglich auftrat

3.2. Konzepte

Eine Programmiersprache ist je nach umgesetzten sprachlichen Konzepten mehr oder weniger gut für die Entwicklung von Web-Anwendungen geeignet. Die Fragen, die man sich jetzt zwangsläufig stellt, sind nun:

- Welche Sprachkonzepte gibt es überhaupt?
- Wie elegant sind die Sprachkonzepte in Java, Ruby und Smalltalk implementiert?

Diese Fragen sollen in den folgenden beiden Abschnitten, auch unter Zuhilfenahme der in Abschnitt 3.1 gemachten Voruntersuchungen beantwortet werden. Im Anschluß daran, findet aufgrund dessen eine Auswahl von Konzepten statt, anhand derer ein Programmiersprachen-Vergleich zwischen Ruby, Java und Smalltalk unter Bezug auf die Referenzanwendung gemacht werden soll.

3.2.1. Überblick

Tabelle 3.1 über Programmiersprachenkonzepte soll als Arbeitsgrundlage für die Auswahl geeigneter Vergleichskonzepte dienen. Sie basiert auf [Voegele].

#	Konzept	Smalltalk	Ruby	Java
1	Objektorientierung	rein	rein	hybrid
2	Kapselung, Geheimnisprinzip	+	+	+
3	Vererbung	einfach	einfach, mehrfach (Mixins)	einfach, mehrfach (Interfaces)

#	Konzept	Smalltalk	Ruby	Java
4	alle vordefinierten Typen sind Objekte	+	+	-
5	alle Operationen sind Nachrichten an Objekte	+	+	-
6	alle anwenderdefinierten Typen sind Objekte	+	+	+
7	Uniform Access	n/a	+	-
8	Generische Klassen	n/a	n/a	+
9	Klassenvariablen- und Methoden	+	+	+
10	Überladung von Methoden	-	-	+
11	Überladung von Operatoren	+	+	-
12	Typisierung	stark, dynamisch	stark, dynamisch (duck)	stark, statisch
13	Compiler/Interpreter	Bytecode/Interpreter	Interpreter	Bytecode
14	Funktionen höherer Ordnung und Lexikalische Closures	+ (Blöcke)	+ (Blöcke)	-
15	Continuations	+	+	-
16	Aliasing	-	+	-
17	Speicherbereinigung	mark and sweep or generational	mark and sweep	mark and sweep or generational
18	Reflection	+	+	+
19	Design by contract	-	add-on	-
20	Multithreading	implementation-dependent	+	+
21	Reguläre Ausdrücke	-	+ (built-in)	+ (stdlib)
22	Zeigerarithmetik	-	-	-
23	Sprachintegration	C	C, C++, Java	C, some C++
24	Built-in Security	-	+	+

#	Konzept	Smalltalk	Ruby	Java
25	Modul- und Klassenbibliotheken	vorhanden	vorhanden	sehr groß

Tabelle 3.1.: Überblick Sprachkonzepte

Ein erster Blick auf Tabelle 3.1 zeigt das Ähnlichkeitsprofil in Tabelle 3.2. Es handelt sich um eine grobe quantitative Betrachtung von Merkmalen, aber sie deutet auf eine stärkere Ähnlichkeit zwischen den Sprachen Ruby und Smalltalk hin.

Ausprägungstyp	Konzeptnummern	Anzahl
alle Sprachen gleiche Ausprägung	2, 6, 9, 18 und 22	5
alle Sprachen unterschiedliche Ausprägungen	3, 7, 13, 21, 23	5
Ruby, Smalltalk gleiche Ausprägung, Java anders	1, 4, 5, 8, 10, 11, 12, 14, 15 und 25	10
Ruby, Java gleiche Ausprägung, Smalltalk anders	24 und 20	2
Java, Smalltalk gleich, Ruby anders	17 16 und 19	3

Tabelle 3.2.: Sprachen-Ähnlichkeitsprofil

3.2.2. Beschreibung und Bewertung der Konzepte

Die in Abschnitt 3.2.1 tabellarisch eingeführten Konzepte sollen hier definiert werden. Im Anschluß an die Beschreibung soll eine Bewertung des Konzeptes hinsichtlich seiner Eignung als Vergleichskriterium stattfinden. Als geeignet erscheint ein Konzept, wenn es im Rahmen der verfolgten Untersuchungs-Strategie (Abschnitt 1.2.2.2) verwendet werden kann. Ein Fazit weist durch ein '+' auf gute Eignung, durch ein '-' auf schlechte Eignung und durch ein 'o' auf eine Eignung mit Einschränkung hin.

3.2.2.1. Objektorientierung

Beschreibung: Es besteht keine einheitliche Meinung darüber, welche Merkmale eine objektorientierte Sprache aufweisen muss. In diesem Kontext seien alle Punkte aus folgender Aufzählung erfüllt, damit sich eine Sprache als rein objektorientiert bezeichnen darf. Eine hybride Sprache wird einigen, aber nicht allen Punkten genügen.

1. Kapselung, Geheimnisprinzip
2. Vererbung
3. Polymorphismus / Dynamische Bindung
4. Alle vordefinierten Typen sind Objekte
5. Alle Operationen sind Nachrichten an Objekte
6. Alle anwenderdefinierten Typen sind Objekte

Bewertung: Eine starke Ausprägung erweist sich als Vorteil, da jede Datenstruktur in der Sprache in einer einheitlichen Weise behandelt werden kann. Die Art der Implementierung von Vererbung in einer Sprache entscheidet auch über den möglichen Einsatzumfang. Eine Sprache kann Mehrfachvererbung unterstützen. Ruby tut dies durch Mixins, Java durch Interfaces und Smalltalk unterstützt nur Einfachvererbung.

Fazit: +

3.2.2.2. Uniform Access

Beschreibung: Es handelt sich um ein Prinzip, welches in [Meyer (1997)] veröffenlicht wurde. Darin heißt es:

> „All services offered by a module should be available through a uniform notation, which does not betray whether they are implemented through storage or through computation."

Weiter heißt es:

> „Although it may at first appear just to address a notational issue, the Uniform Access principle is in fact a design rule which influences many aspects of object-oriented design and the supporting notation. It follows from the Continuity criterion; you may also view it as a special case of Information Hiding."

Bewertung: „Uniform Access" ist eine vorteilhafte Eigenschaft, die das Geheimnisprinzip der Objektorientierung unterstützt.

Fazit: +

3.2.2.3. Generische Klassen

Beschreibung: Bei vorhandener Ausprägung hat man in einer statisch typisierten Sprache die Möglichkeit eine Datenstruktur mit einem Datentyp zu parametrisieren. Eine Stack-Datenstruktur beispielsweise kann so jede Art von Objekten aufnehmen und ist gleichzeitig typsicher zur Kompilierzeit.

Bewertung: Java verfügt über dieses Konzept seit Version 5.0. Ruby und Smalltalk benötigen es nicht, da Typen erst zur Laufzeit geprüft werden. Generizität fällt bei diesen Sprachen zusammen mit Reflection als Nebenprodukt ab. Ein gut durchdachtes Typsystem ist im Falle von statischen Sprachen wie Java wichtig, um weniger Typfehler zu generieren. Java kämpft hier mit Problemen, wie in Abschnitt 3.1.2 angeführt. Ein Vergleich in diesem Punkt kann sich lohnen.

Fazit: +

3.2.2.4. Klassenvariablen- und Methoden

Beschreibung: Klassenvariablen und -methoden sind Bestandteil einer Klasse und nicht von Objekten dieser Klasse. Daher existiert zu jedem Zeitpunkt nur eine Kopie der jeweiligen KVM, welche von allen Objekten geteilt werden. In Smalltalk und Ruby sind KVM sehr natürlich integriert aufgrund der Tatsache, daß bei ihnen selbst Klassen Objekte sind. Java liefert durch „static members“ das gleiche, jedoch ohne die Möglichkeit diese KVM zu vererben.

Bewertung: Das Konzept ist nicht geeignet, da es von allen Sprachen hinreichend elegant erfüllt wird.

Fazit: -

3.2.2.5. Überladung von Methoden

Beschreibung: Bezeichnet die Fähigkeit einer Klasse oder eines Moduls zwei oder mehrere Methoden mit demselben Namen zu haben. Unterschieden werden diese nur durch die Art und die Anzahl der Parameter.

Bewertung: Aufgrund der statischen Typisierung muss in Java das Überladen beim Aufbau einer API exzessiv genutzt werden, um API Support für mehrere Typen zu gewährleisten. Für die Klasse java.util.Array beispielsweise führt das zu 70 Methoden mit nur zehn verschiedenen Funktionalitäten. Ruby bedient sich eines anderen Ansatzes: Für Methoden, bei denen Überladung genutzt wird liefern Standard-Argumentwerte wie `0` oder `null` ein vordefiniertes Standardverhalten. Methoden-Überladung führt zu einer aufgeblähten API. Dies wirkt sich jedoch nicht ausgesprochen hinderlich auf den Versuch aus, elegant zu codieren, da bei differierenden Parameter-Datentypen automatisch die richtige Methode ausgewählt wird.

Fazit: -

3.2.2.6. Überladung von Operatoren

Beschreibung: Bezeichnet die Möglichkeit für den Programmierer einen Operator für eine Funktion zu definieren. Auf diese Weise kann die Funktion als infix-, prefix- oder postfix-Operator ausgeführt werden. In einem mathematisch-numerischen Kontext erweist sich diese Möglichkeit als sehr nützlich, da mathematische Formeln im Code auf natürliche Weise festgehalten werden können.

Bewertung: Für Entwickler mathematischer Bibliotheken ist das ein nützliches Konzept. Für die fachliche Anwendungsentwicklung ist es nicht so ausschlaggebend.

Fazit: -

3.2.2.7. Typisierung

Beschreibung: Es gibt zwei Begriffspaare im Zusammenhang mit der Typisierung einer Sprache. „Starke und Schwache“ sowie „Statische und Dynamische“ Typisierung. In diesem Kontext gelten folgende Definitionen:

Starke versus Schwache Typisierung: Eine Sprache ist stark typisiert, wenn sie für jeden Datentyp nur typkompatible Operationen zuläßt. In Ruby, einer stark typisierten Sprache, käme es beispielsweise zu einem „Type Error“ bei dem Versuch einen String mit einer Zahl zu addieren. In C, einer schwach typisierten Sprache, ist es ohne weiteres möglich ein Char mit einem Integer zu addieren, da C intern eine Umwandlung des Chars in seinen Integer-ASCII-Code vornehmen würde und so im Ergebnis ein Integer liefern würde.

Statische versus Dynamische Typisierung: Statisch typisierte Sprachen fordern, daß jede Variable, jeder Funktionsparameter und jeder Funktionsrückgabewert spätestens zur Kompilierzeit mit einem Datentyp versehen wurden. In dynamisch typisierten Sprachen entscheidet sich erst zur Laufzeit, welchen Datentyp eine Variable hat. Eine andere Formulierung lautet: Statische Typisierung bindet einen Typ an eine Variable, Dynamische Typisierung bindet einen Typ an ein Objekt.

Bewertung: Starke und Schwache Typisierug sind in dieser Betrachtung weniger interessant. Schwache Typisierung ist manchmal komfortabler, aber auch weniger vorhersagbar, da es schlicht keine Typfehler abfängt. Java bietet ein Schlupfloch in Richtung schwache Typisierung, indem es das Casten von Objekten erlaubt. Wichtiger scheint hier eher die Unterscheidung zwischen Statischer und Dynamischer Typisierung zu sein. Fürsprecher der statischen Typisierung behaupten, daß durch sie sicherer, verläßlicherer Code erzeugt wird und die Effizienz des Endprodukts erhöht wird. Liebhaber dynamischer Sprachen unterstreichen die erhöhte Flexibilität und Produktivität. In jedem Falle benötigt eine statisch typisierte Sprache ein ausgeklügeltes Typsystem, um so flexibel wie seine dynamischen Konkurrenten zu sein. Aufgrund der Vorarbeiten in Abschnitt 3.1.2 wurde ersichtlich, daß Java hier mit Schwierigkeiten zu kämpfen hat. Dynamisch typisierte Sprachen haben auf den ersten Blick ein Typsicherheitsproblem. Wenn eine Variable ein Objekt enthält, das man dort nicht erwartet hat, muß man mit einer Art von „message not understood“-Fehler rechnen. Durch ausführliches Unit-Testing ist diese Problematik jedoch marginal. In einer statischen Sprache kann dieser Fehler nicht auftreten.

Ein statisches Typsystem führt zu drei weiteren Problemen:

- Erstens zwingt es den Programmierer zu verfrühter Detailarbeit.
- Zweitens wird das Experimentieren mit der Sprache erschwert
- Drittens wird die Pflege und Wartung von Software erschwert.

Punkt eins und zwei sind offensichtlich: Man muß sich zu jeder Zeit zu Fragen der verwendeten Datentypen festlegen und exploratives Kennenlernen der Sprache wird durch den ständigen Zwang alles zu typisieren erschwert. Für Punkt drei stelle man sich vor, daß eine Methode ein bestimmtes Objekt als Paramter annimmt. Sollte sich der Typ des Objekts im Zuge der Pflege ändern, muss auch die Signatur der Methode geändert werden. Wenn dasselbe Objekt auch an andere Methoden übergeben wurde, müssen diese auch entsprechend angepaßt werden. Man erkennt eine Tendenz bei statischen Sprachen zu erhöhter Kopplungsgefahr. In einer dynamischen Sprache müßte man im Falle einer Änderung des Typs eines Funktionsparamters lediglich dafür sorgen, daß der neue Typ das alte Methodenprotokoll versteht, was im schlimmsten Fall eine Änderung an einer Stelle des Codes nach sich zieht. Die Art der Typisierung hat also großen Einfluß auf elegante Codierung. Sie kann daher zur Untersuchung herangezogen werden.

Fazit: +

3.2.2.8. Compiler/Interpreter

Beschreibung: Eine Sprache wird grob gesagt entweder interpretiert oder kompiliert. Im ersten Fall wird der Programmquellcode nicht wie beim Kompilieren in eine auf dem System direkt ausführbare Datei umgewandelt, sondern eingelesen, analysiert und ausgeführt. Die Analyse des Quellcodes erfolgt also zur Laufzeit des Programms.

Bewertung: Der größte Nachteil der Interpretersprachen ist die im Vergleich zu compilierten Programmen deutlich langsamere Ausführungsgeschwindigkeit. Für die Webentwicklung ist dies jedoch zweitrangig, da HTTP-Anfragen meist keine rechenintensiven Berechnungen nach sich ziehen. Viel wichtiger ist hier der Umstand hervorzuhaben, daß sich bei interpretierten Sprachen der „Code Compile Cycle“ deutlich beschleunigt, was in der Anwendungsentwicklung für ein schnelles Feedback von gemachten Änderungen im Code sorgt. Ruby und Smalltalk sind interpretierte Sprachen, Java muß bytecode-kompiliert werden.

Fazit: o

3.2.2.9. Funktionen höherer Ordnung und Lexikalische Closures

Beschreibung: FHO sind Funktionen, die wie Datenobjekte behandelt und somit an Variablen gebunden werden können. Infolgedessen können sie auch anderen Funktionen als Parameter übergeben oder von diesen als Wert zurückgeliefert werden. Die Ausführung von Code in FHO kann damit verzögert werden, was zur Folge hat, daß sie je nach Aufrufort einen anderen Ausführungskontext[9] vorfindet. LC gehen noch einen Schritt weiter. Sie speichern den Kontext an ihrem Definitionsort, sodaß sie stets Zugriff auf ihre ursprüngliche Umgebung haben, gleichgültig wo sie aufgerufen werden und gleichgültig ob der ursprüngliche Definitionskontext zum Zeitpunkt des Aufrufs noch exisitiert oder nicht. Smalltalk und Ruby unterstützen beide Konzepte in Form von Blöcken. Java bietet ein Konstrukt namens „Anonyme Klasse“ an, in dem auch Funktionen zusammengefaßt werden können. Dadurch können FHO annähernd simuliert werden, da diese Funktionen auch an Variablen gebunden und auch als Parameter an andere Funktionen übergeben und zurückgeliefert werden können. Jedoch sind diese Funktion benannt und können daher nicht in der gleichen generischen Weise verwendet werden wie wirkliche FHO. LC werden von Java gar nicht unterstützt.

[9] Variablen und deren Belegungen

Bewertung: LC erhöhen die Ausdruckskraft und Mächtigkeit einer Sprache. Es ergeben sich zum Beispiel folgende Einsatzmöglichkeiten [Wikipedia (2007e)]:

- Code-Anpassung: Entwickler von Softwarebibliotheken können Nutzern die Möglichkeit bieten das Verhalten von Bibliotheksfunktionen durch Übergabe von LC als Argumente anzupassen. Zum Beispiel könnte eine Sortier-Funktion eine LC akzeptieren, die die Art des Vergleichs bei der Sortierung festlegt.
- Kontrollstrukturen: LC können als Kontrollstrukturen verwendet werden. In Smalltalk werden alle Kontrollstrukturen und Schleifen definiert, indem Objekte genutzt werden, deren Methoden Blöcke als Argumente akzeptieren. Ein if-then-else-Konstrukt nimmt damit die Form in Listing 3.11 an.
- Datenpersistenz über mehrere Funktionsaufrufe: Mehrere Funktionen können durch LC auf eine gleiche Umgebung zugreifen. Dadurch kann eine private Kommunikation durch gemeinsames Ändern der LC stattfinden.

```
aBoolean ifTrue: [ ... ] ifFalse: [ ... ]
```

Quellcode 3.11: Kontrollstrukturen in Smalltalk

Fazit: +

3.2.2.10. Continuations

Beschreibung: Nach Wikipedia (2007g) sind Continuations ein Punkt in der Ausführung eines Programms, also der Befehlszähler zusammen mit dem dazugehörigen „Stack Frame“. Realisiert eine Programmiersprache dieses Konzept, dann kann zur Laufzeit eine Continuation erzeugt und der Programmablauf später an diesem Punkt wieder aufgenommen werden. Continuations können je nach Ausprägung mehrmals oder wie in Ruby einmal aufgerufen werden.

Bewertung: In der Webprogrammierung gewinnen Continuations an Bedeutung, da der asynchrone Anfrage/Antwort-Zyklus durch sie linearisiert werden kann. Ein Geschäftsprozess, der nur durch mehrere HTTP-Anfragen und -Antworten abgewickelt werden kann, kann nun innherhalb einer Methode codiert werden. Nach jeder Antwort wird die aktuelle Continuation gespeichert und bei einer erneuten Anfrage desselben Nutzers wieder geladen. Hinter dem alten Ausstiegspunkt wird dann weiter fortgefahren.[10]

[10] ein ausführlich beschriebenes Beispiel zur Syntax und Funktionsweise von Continuations in Ruby kann Seite 143 aus [Tate (2005)] entnommen werden

Fazit: +

3.2.2.11. Aliasing

Beschreibung: Eine Möglichkeit Attribute oder Methoden einer Klasse umzubenennen.

Bewertung: Dieses einfache Konzept wird nur von Ruby unterstützt. Mit seiner Hilfe ist es möglich das Verhalten einer Klasse zu ändern ohne deren Code zu ändern. Für ein Code-Beispiel werfe man einen Blick auf Listing 3.6. AOP[11] läßt sich mithilfe von Aliasing implementieren [Tate (2005)].

Fazit: +

3.2.2.12. Speicherbereinigung

Beschreibung: Die Speicherbereinigung besorgt das automatische Reinigen von Speicher von Objekten, die nicht mehr benutzt werden. Die Alternative liegt in der händischen Befreiung von ungenutztem Speicher durch den Programmierer.

Bewertung: Hinsichtlich der Möglichkeit Anforderungen elegant zu implementieren, ist dieses Konzept nicht ausschlaggebend, da es im Hintergrund abläuft.

Fazit: -

3.2.2.13. Reflection

Beschreibung: Reflection ist die Fähigkeit einer Sprache zahlreiche Informationen über ein Objekt zur Programm-Laufzeit herauszufinden. Also den Typ eines Objekts, seine Stelle in der Vererbungshierarchie, seine Methoden inklusive Anzahl und Typ der Parameter und Rückgabewerte. Weiterhin kann Reflection auch die Fähigkeit beinhalten die Namen und Typen der Attribute herauszufinden.

[11] siehe Abschnitt 5.2.1

Bewertung: Die meisten objektorientierten Sprachen unterstützen eine Form von Reflection und ermöglichen damit die Metaprogrammierung, also die Erzeugung von Programmcode durch Programmcode. Metaprogrammierung kann sich als mächtiges Werkzeug erweisen, da es unter anderem eine komfortable Möglichkeit bietet „Domänenspezifische Programmiersprachen“ zu implementieren [Schempp (2006)]. Dieses Konzept wird daher als wichtiges Untersuchungskriterium erachtet. Nach Tabelle 3.1 unterstützen zwar alle betrachteten Sprachen dieses Konzept, jedoch bleibt abzuwarten, was der Sprachvergleich in Abschnitt 4 in Hinblick auf die Konsequenz und Eleganz der Umsetzung zutage fördert.

Fazit: +

3.2.2.14. Design by contract

Beschreibung: „Design by contract“ [Wikipedia (2007j)] ist ein von Bertrand Meyer in [Meyer (1997)] vorgestelltes Konzept, welches ein korrektes Zusammenspiel von Programmodulen durch die Definition formaler Verträge zwischen ihnen garantieren kann. Diese Idee basiert auf einer Metapher aus dem Geschäftsleben. Dort gibt es „Kunden“ und „Anbieter“, die der gegenseitigen Erfüllung eines „Vertrags“ zustimmen. Dieser definiert die zu erbringenden Leistungen und Gegenleistungen beider Parteien durch „Verpflichtungen“ und „Vorteile“ sowie Verpflichtungen, die zu jeder Zeit erfüllt sein müssen, wie zum Beispiel Handelsgesetze.

Bei Verwendung einer Terminolgie aus der Softwareentwicklung, stellt sich das Konzept so dar:

- Ein Kunden-Modul verpflichtet sich beim Aufruf eines Anbieter-Moduls dessen „Vorbedingungen“[12] einzuhalten. Das beinhaltet beispielsweise, daß Aufrufparameter innerhalb bestimmter Grenzen liegen. Das Anbieter-Modul hat den Vorteil keine Fälle behandeln zu müssen, die sich außerhalb der Vorbedingung befinden.
- Das Anbieter-Modul verpflichtet sich nach korrekter Einhaltung seiner Vorbedinungen durch das Kunden-Modul eine Aktion auszuführen, welcher seinen zuvor vertraglich festgelegten „Nachbedingungen“[13] genügt. Das Kunden-Modul hat den Vorteil, daß es sich sicher sein kann einen Wert zu erhalten, der in den Nachbedingungen fesgelegt wurde.
- Das Anbieter-Modul garantiert außerdem, daß bestimmte Eigenschften, vor und nach seinem Aufruf, erhalten bleiben. Dies ist in seiner „Invariante“ festgelegt.

Durch Vor-, Nachbedingungen und Invarianten werden in einem konkreten Vertrag für eine Methode die Wertebereiche der Eingangsparameter genau klassifiziert, sowie deren Randbedingungen und Anforderungen festgehalten. Dementsprechend werden auch die Ausgangspa-

[12] engl. preconditions
[13] engl. postconditions

rameter beschrieben. Es wird also exakt definiert, in welcher Situation eine Methode welche Werte verarbeiten kann. Auf diese Weise wird Korrektheit bei der Zusammenarbeit von Objekten garantiert. Wenn die Invarianten und die Vorbedingungen vor Benutzung eines Dienstes wahr sind, dann sind es die Invarianten und Nachbedingungen auch danach. DBC geht über Sicherungsverfahren, die in statisch typisierten Sprachen durch Typkontrolle gegeben wird, hinaus, da auch Forderungen wie „vor Routine A muss Routine B aufgerufen worden sein“ oder „Routine A ruft in ihrem Verlauf immer auch Routine B auf“ durch Nachbedingungen und Invarianten erfaßt werden können [Wikipedia (2007i)]. In dynamisch typisierten Sprachen kann die Erfüllung von Vor-, Nachbedingungen und Invarianten durch ausgiebiges „Unit-Testing“[14] und „Integration-Testing“[15] erzielt werden. DBC hebt dieses Konzept jedoch auf eine systematische Ebene, da jedes Modul vertraglich spezifiziert wird.

Vorteile [Wikipedia (2007j)]:

- Wiederverwendung von Code wird gefördert, da jedes Modul hinsichtlich seines Verhaltens umfassend dokumentiert ist.
- Testcode im Anbieter-Modul bezüglich von Eingangswerten kann entfallen, was zu einem Performance-Gewinn führt.
- Code-Debugging wird vereinfacht, da gleich zu Beginn klar ist, was das Modul leisten soll.

Bewertung: Das Konzept liefert die genannten Vorteile: Erhöhte Wiederverwendung, Performance-Steigerung und vereinfachtes Debugging. Das Konzept wird zur Zeit nur von Ruby durch Hinzunahme einer Bibliothek unterstützt. Ein Vergleich kann daher nicht stattfinden.

Fazit: +

3.2.2.15. Multithreading

Beschreibung: Beschreibt die Fähigkeit eines Prozesses mehrere Aufgaben nebenläufig auszuführen.

[14] testet einzelne Module
[15] testet das Zusammenspiel von Modulen

Bewertung: Multithreading wird von allen betrachteten Sprachen bereitgestellt. Bei der Web-Anwendungs-Entwicklung wird dieser Aspekt meist in Bibliotheken gekapselt, sodaß der Web-Anwendungs-Entwickler kaum damit in Berührung kommt.

Fazit: -

3.2.2.16. Reguläre Ausdrücke

Beschreibung: „Reguläre Ausdrücke“ sind Mustererkennungs-Konstrukte, mit deren Hilfe festegestellt werden kann, ob „Wörter“ Bestandteil einer „Regulären Sprache“ sind. Sie kommen häufig bei textverarbeitenden Aufgaben zum Einsatz, also bei Text-Such- und -Ersetzungsaufgaben. Java unterstützt Reguläre Ausdrücke durch Hinzunahme von Bibliotheken. Ruby bietet native Unterstützung für RA an. Smalltalk gar keine Unterstützung.

Bewertung: Dieses Kriterium ist geeignet für die Untersuchung. Eine gelungene Integration von RA in die Programmiersprache ermöglicht eine einfache und komfortable Benutzung dieser Konstrukte. String-Verarbeitung ist eine sehr häufige Aufgabe bei der Entwicklung von Web-Anwendungen.

Fazit: +

3.2.2.17. Zeigerarithmetik

Beschreibung: „Zeigerarithmetik“ bezeichnet die Fähigkeit einer Sprache Speicheradressen und ihre Inhalte direkt zu manipulieren. Keine der betrachteten Programmiersprachen unterstützt ZA.

Bewertung: Die hier betrachteten Programmiersprachen sind Hochsprachen, welche eine gute Unterstützung bei der Entwicklung fachlich motivierter Anwendungen bieten sollen und daher von der Last befreit sein sollen Speicher zu reservieren und freizugeben und Speichergrenzen zu berücksichten. Im Gegensatz dazu stehen Sprachen für die Systemprogrammierung, mit denen beispielsweise Betriebssysteme programmiert werden können. Diese Sprachen brauchen Fähigkeiten zur direkten Speichermanipulation, da sie auch Aufgaben auf Hardwarebene haben. Web-Anwendungs-Entwickler kommen mit diesem Konzept nicht in Berührung. Ein Sprachen-Vergleich erübrigt sich deswegen.

Fazit: -

3.2.2.18. Sprachintegration

Beschreibung: Die Fähigkeit einer Sprache Code von „low level“-Sprachen aufzurufen und zu benutzen.

Bewertung: Ruby, Smalltalk und Java können alle mit der System-Programmiersprache C interagieren. Die Integration von performancekritischem Code in hardwarenahen Sprachen wie C führt zu Geschwindigkeitsgewinn. Die Beschäftigung damit fällt jedoch nicht in den Bereich der Web-Anwendungs-Entwicklung.

Fazit: -

3.2.2.19. Built-in Security

Beschreibung: „Built-in Security“ ist die Fähigkeit einer Sprache zu entscheiden, ob Code aus einer vertrauenswürdigen Quelle kommt.[16] Im gegenteiligen Fall muss die Sprache in der Lage sein die Zugriffs-Rechte dieses Codes zu limitieren. Ein Beispiel: Java-Applets werden als nicht vertrauenswürdig erachtet. Daher sind sie in den Aktionen, die sie ausführen dürfen, beschränkt, wenn sie innerhalb des Browsers ausgeführt werden. Sie dürfen beispielsweise nicht von der Festplatte des Anwenders lesen oder auf sie schreiben. Sie dürfen auch keine Netzwerkverbindung aufbauen außer zum ursprünglichen Host. Java bietet diese Sicherheit in hervorragender Weise durch seine JVM. Ruby bietet diese Fähigkeit. Smalltalk nicht.

Bewertung: BIS wird auf einer Ebene unterhalb der Web-Anwendungs-Entwicklung implementiert. In Java durch die JVM, Ruby bietet sie, Smalltalk nicht [Voegele]

Fazit: -

3.2.2.20. Modul- und Klassenbibliotheken

Beschreibung: Ein großer Umfang an Modul- und Klassenbibliotheken spricht für die Verwendung einer Sprache.

[16] als Beispiel sei hier die Festplatte genannt

Bewertung: Dieses Kriterium verdient Beachtung. Eine Sprache kann von dem Reichtum der Bibliotheken einer anderen profitieren, wenn es diese in irgendeiner Weise integrieren kann.

Fazit: +

3.2.3. Auswahl relevanter Konzepte

Tabelle 3.3 ist das Ergebnis der Untersuchung aus Abschnitt 3.2.2 und zeigt für jedes Sprachkonzept seine Eignung als Untersuchungskriterium zur Sprach-Eleganz an. Es sollten bei den mit einem '+' markierten Konzepten signifikante Unterschiede feststellbar sein, anhand derer sich Aussagen über die Produktivität und die Eignung für die Webprogrammierung der untersuchten Sprachen ableiten lassen.

#	Konzept	Eignung
1	Objektorientierung	+
7	Uniform Access	+
14	Funktionen höherer Ordnung und Lexikalische Closures	+
15	Continuations	+
16	Aliasing	+
12	Typisierung	+
18	Reflection	+
21	Reguläre Ausdrücke	+
25	Modul- und Klassenbibliotheken	+
8	Generische Klassen	+
19	Design by contract	+
13	Compiler/Interpreter	o
24	Built-in Security	-
9	Klassenvariablen- und Methoden	-
10	Überladung von Methoden	-
11	Überladung von Operatoren	-
17	Speicherbereinigung	-
20	Multithreading	-
22	Zeigerarithmetik	-
23	Sprachintegration	-

Tabelle 3.3.: Sprach-Konzepte und ihre Eignung als Untersuchungskonzepte

Im Rahmen dieser Arbeit sollen folgende Konzepte unter Rückbezug auf die Referenzanwendung untersucht werden.

- Objektorientierung/Vererbung
- Funktionen höherer Ordnung und Lexikalische Closures
- Generische Klassen
- Reflection

4. Sprachvergleich am Beispiel

4.1. Objektorientierung - Vererbung

Alle untersuchten Sprachen sind objektorientiert und bieten daher das Konzept der einfachen Vererbung an. Eine Unterstützung für Mehrfachvererbung wäre wünschenswert. Ruby bietet dafür die Mixins an, Java unterstützt das Konzept durch Interfaces. Smalltalk bietet keine Unterstützung für Mehrfachvererbung an [Wikipedia (2007u)]. Als Beispiel aus dem GISTA sei das CRUD-Plugin herangezogen. Eine Controller-Klasse, zum Beispiel der `UsersController` wird zur Laufzeit um CRUD-Zugriffsmethoden erweitert, indem allen Controllern per `include` Anweisung das Modul `CRUD::ClassMethodes` hinzugefügt wird.[17] Zu Demonstrationszwecken wird in den folgenden Beispielen die `include` Anweisung im Controller angegeben. Die Ruby-Variante ist in Code-Beispiel 4.1, die Java-Variante in Code-Beispiel 4.2.

Bewertung: Anhand der Code-Beispiele ist zunächst kein großer Unterschied hinsichlich des Code-Umfangs ersichtlich. Bei der Java-Variante müssen jedoch alle Interface-Methoden für jeden Controller implementiert werden. Bei Ruby sind sie durch den Mixin-Mechanismus sofort in jedem Controller verfügbar.

```ruby
class UsersController < ApplicationController
  include CRUD::ClassMethods
  # ...
end
```

Quellcode 4.1: Ruby Mehrfachvererbung

```
public class UsersController extends ApplicationController
  implements CRUD.ClassMethods
{
  // ...
```

[17] siehe dazu auch Abschnitt 4.4

```
}
```

Quellcode 4.2: Java Reguläre Ausdrücke

4.2. Funktionen höherer Ordnung und Lexikalische Closures

Iteration in Java, Ruby, Smalltalk: Beim GISTA gibt es eine Funktion, um alle Nutzer mit Admin-Status als Array zurückzuliefern. Dies funktioniert mittels Block wie in Codebeispiel 4.3. Die Nutzung von Blöcken in Ruby und in Smalltak (Codebeispiel 4.5) reduziert die Anzahl der notwendigen Zeilen. Die Java-Variante (Codebeispiel 4.4) braucht mehr als doppelt soviel Zeilen an Code.

```
def admins(users=[])
  users.select {|e| e.isAdmin}
end
```

Quellcode 4.3: Ruby Iteration

```
public static IList Admins(IList emps)
{
  IList result = new ArrayList();
  foreach(User u in users)
    if (u.IsAdmin) result.Add(e);
  return result;
}
```

Quellcode 4.4: Java Iteration

```
admins: users
 ^users select: [:e| e isAdmin]
```

Quellcode 4.5: Smalltalk Iteration

Code-Anpassung: Die Übergabe von Code als Closure wird in Smalltalk und Ruby mittels Blöcken gehandhabt. In Java wird dafür eine anonyme Klasse verwendet. Im GISTA gibt es eine Methode im User-Model, mit der man den Vor- und Nachnamen eine Nutzers zurückgeben kann. Durch Angabe eines Blocks kann man die Art und Weise, wie beide Bestandteile des Namens zurückgegeben werden nach Belieben anpassen. Zum Beispiel könnte man

statt <Vorname> <Nachname> auch <Nachname>, <Vorname> ausgeben. In den assozierten Codebeispielen wird die letzte Variante gezeigt. Codebeispiel 4.6 zeigt die Ruby-Variante, Codebeispiel 4.7 die Java-Variante und Code-Beispiel 4.8 die Smalltalk-Variante.

Bewertung: In Smalltalk und Ruby ist die Code-Anpassung durch Blöcke mit einer Zeile Code implementiert. In Java benötigt man ein Interface, Blöcke werden durch dieses Interface implementierende anonyme Klassen simuliert. Alles in allem zählt man bei der Java-Variante 15 Zeilen Code, bei der Ruby Variante drei Zeilen.

```ruby
def fullName(&block)
  block.call(self.first_name, self.last_name)
end

# **** Aufuf ******
fullName {|a,b| b+', '+a}
```

Quellcode 4.6: Ruby Codeanpassung

```java
interface SFromSAndS
{
   String call(String a, String b);
}

// Anonyme Klasse als Block-Äquivalent
SFromSAndS concatNames = new SFromSAndS()
{
   public String call(final String a, final String b)
   {
      return b + ", " + a;
   }
};

// ...

public String fullName(SFromSAndS s)
{
 s.call(this.first_name, this.last_name)
}

// ...
//**** Aufuf ******
```

```
fullName(concatNames);
```

Quellcode 4.7: Java Codeanpassung

```
fullName: block
  ^block value: self.first_name value: self.last_name

"..."
" **** Aufuf ****** "
fullName [:a :b| b+', '+a]
```

Quellcode 4.8: Smalltalk Codeanpassung

4.3. Generische Klassen

Für das GISTA soll es in einem `Route` Model[18] die Möglichkeit geben unter anderem das Ziel einer Route in einer Übersicht darzustellen. Dazu muß dem Route-Array das letzte Element entnommen werden. Code-Beispiel 4.9 zeigt die Ruby-Variante, Code-Beispiel 4.10 die Java-Variante und Code-Beispiel 4.11 die Smalltalk-Variante.

```
def ziel
  self.last
end
```

Quellcode 4.9: Ruby Variante Routenziel

```
public Point ziel() {
 Route r = this;
 Point p = (Point)r.get(r.length-1);
 return p;
}
```

Quellcode 4.10: Java Variante Routenziel

```
ziel
```

[18] stellt im Objektmodell die Routen dar

```
self last
```

Quellcode 4.11: Smalltalk Variante Routenziel

4.4. Reflection

Controller in Ruby on Rails übernehmen in jeder Anwendung die Aufgabe zwecks Geschäftslogik-Berechungen meist auf das ihnen zugeordnete Model zuzugreifen, indem sie Model-Instanzen lesen, erstellen, verändern oder löschen. Dazu müssen in ihnen entsprechende CRUD-Zugriffsmethoden definiert sein. Anstatt diese jedesmal für jeden Controller per Hand zu erstellen, kann man diese auch erst zur Laufzeit des Programms generieren lassen. Dies setzt Reflection-Fähigkeiten einer Sprache voraus. Auch in der GISTA-Anwendung ist ein Plugin integriert worden, welches diese Aufgabe übernimmt. Im `vendor/plugins/acts_as_crud` Verzeichnis befinden sich alle relevanten Module. Unter anderem die Datei `init.rb`. Diese wird bei jedem Applikations-Start ausgeführt.[19] Dort werden dann allen Controllern wie in Codebeispiel 4.12 per inlcude (Mixin) alle Zugriffsmethoden zur Laufzeit hinzugefügt.[20] Die adäquate Implementierung in Java zeigt Codebeispiel 4.13[21], die für Smalltalk zeigt Codebeispiel 4.14[22].

```
ActionController::Base.class.send(:include, CRUD::ClassMethods)
```

Quellcode 4.12: Ruby Reflection

```
// ...
Class c = Module.class;
Class[] parameterTypes = new Class[] {String.class};
Method includeMethod;
Object[] arguments = new Object[] {"CRUD.ClassMethods"};
try
{
  includeMethod = c.getMethod("include", parameterTypes);
  includeMethod.invoke("ActionController.Base", arguments);
```

[19] für eine genaue Beschreibung zum Aufbau und zur Erstellung von Rails-Plugins siehe [Grosenbach]

[20] Vereinfacht dargestellt. Für die genaue Implementierung siehe Anhang

[21] die Java-Reflection-API-Dokumentation auf [von www.javaCore.de] wurde als Grundlage für das Beispiel genommen. Man muß auch bedenken, daß es in Java keine include-Methode für Objekte gibt. Das Beispiel ist jedoch geeignet den Unterschied im syntaktischen Aufwand zu verdeutlichen.

[22] Smalltalk-Beispiel basiert auf [Wikipedia (2007r)]

```
}
catch (NoSuchMethodException e)
{
    System.out.println(e);
}
catch (IllegalAccessException e)
{
    System.out.println(e);
}
catch (InvocationTargetException e)
{
    System.out.println(e);
}
//...
```

Quellcode 4.13: Java Reflection

```
(Compiler evaluate: 'ActionController>>Base') perform: #include with: 'CRUD>>Cl
```

Quellcode 4.14: Smalltalk Reflection

4.5. Fazit

In Ruby und Smalltalk läßt sich Code im Gegensatz zu Java durch angebotene Konzepte wie Closures durch Blöcke, die dynamische Typisierung und gut integriertes Reflection wesentlich eleganter formulieren. Der Code wirkt dort wie prosaisches Englisch. Er ist daher verständlicher und durch die Knappheit leichter zu warten und weiterzuentwickeln. Ruby kann durch sein Mixin-Konzept Mehrfachvererbung leicht und elegant implementieren ohne auf einschlägige Schwierigkeiten zu stoßen [Wikipedia (2007k)].

In Abschnitt 4 wurden die betrachteten Programmiersprachen anhand der in Abschnitt 3.2.3 ausgewählten Konzepte untersucht und verglichen. Es handelt sich um einen technischen nur auf die Sprachkonzepte bezogenen Vergleich. Im folgenden Abschnitt 5 sollen nun auch die hervorstechendsten Merkmale der zu den Sprachen gehörenden für die Webentwicklung gedachten Frameworks untersucht werden. Besonderes Augenmerk soll beim Vergleich und der Bewertung dieses Folgeabschnitts auf die Frage gelegt werden, ob die Frameworks in irgendeiner Weise von den Implementierungen der Spracheigenschaften profitieren oder im Gegenteil Nachteile aus ihnen ziehen.

5. Webentwicklungsframeworks

5.1. Allgemein

Webentwicklungsframeworks sind Frameworks, die einen bei der Entwicklung von Webanwendungen unterstützen. Sich ständig wiederholende, domänenspezifische Aufgaben sollen mit ihrer Hilfe vereinfacht und verringert werden. Solche Aufgaben beziehen sind auf Aspekte in Tabelle 5.1 [Wikipedia (2007v)][Wikipedia (2007f)].

Zur Bewältigung dieser Aufgaben gibt es in diversen Programmiersprachen Frameworks, welche eine oder mehrere der oben genannten Aspekte abdecken, sich aber vor allem bei Java in ihren gebotenen Leistungen überschneiden [Wikipedia (2007q)]. Interessant sind in diesem Zusammenhang vor allem die „Full Stack“-Frameworks. Das ist eine kohärente Kombination aus ein oder mehreren Frameworks, mit denen eine Webapplikation komfortabel gebaut werden kann. Für Ruby ist „Ruby on Rails“ ein Full-Stack-Framework. Für Smalltalk ist es „Seaside“. Für Java ist es zum Beispiel neben vielen anderen Möglichkeiten die Kombination aus einem JavaEE Application Server wie JBoss, dem Spring-Framework und dem OR-Mapper Hibernate.

5.2. Besondere Paradigmen und Techniken

Es gibt moderne Paradigmen und Techniken, die im Zusammenhang mit der Programmierung von Web-Frameworks eine besondere Rolle spielen. Diese sollen hier kurz erläutert werden, weil bei der Bewertung der betrachteten Frameworks auf sie Bezug genommen wird.

5.2.1. Aspektorientierte Programmierung

Hinter dem AOP-Programmierparadigma steckt der Wunsch querschneidende Software-Anforderungen zu modularisieren, um sie besser warten und wiederverwenden zu können. Software-Anforderungen lassen sich in zwei Kategorien einteilen:

1. Core-Level-Concerns: Fachliche Kernfunktionalitäten, die sich gut kapseln lassen.

Aspekt	Beschreibung
MVC	Trennung des Programm-Codes in Datenmodelle, Geschäftslogik und Anwenderschnittstelle nach dem MVC-Entwurfsmuster
ORM	Datenbankunabhängige Persistierung über eine einheitliche API
Templating	Bereitstellen einer Template-Engine
Sicherheit	Anwender-Authentifizierung und -Authorisierung
Caching	Web-Caching
Ajax	Ajax-Unterstützung
Automatische Konfiguration	Durch Reflection: Zum Beispiel können auf diese Weise zur Laufzeit Datenmodell-Objekte mit Zugriffsmethoden versehen werden, die den zugehörigen Datenbanktabellen-Attributen entsprechen. Ein Datenbankschema braucht daher auf Applikationsebene nicht explizit definiert werden.
Web-Service	Web-Service-Unterstützung
URL mapping	Umschreiben der URL für bessere Lesbarkeit und Suchmaschinenoptimierung
Validierung	Unterstützung für Eingabe-Validierung
i18n und l10n	Internationalisierung und Lokalisierung
AOP	Programmierung nach dem AOP-Paradigma
IoC	Programmierung nach dem IoC-Paradigma
Transaktion	Transaktionsmanagment
Testen	Unterstützung fürs Testen

Tabelle 5.1.: Aufgaben von Webframeworks

2. System-Level-Concerns: Anforderungen, die von mehreren Modulen parallel benötigt werden, wie zum Beispiel Logging.

Core-Level-Concerns kann man als Komponenten bezeichnen, System-Level-Concerns sind die „Aspekte“. Erstere kann man mittels Modulen oder Klassen implementieren, zweitere sind ohne weiteres nicht zu kapseln, sondern sind quer über alle Komponenten verteilt.

Beispiel: Der Ablauf eines Programms soll mitverfolgt werden (Tracing). Dazu soll bei Ein- und Austritt aus einer Funktion eine Meldung geschrieben werden wie in Codebeispiel 5.1 ersichtlich. Problem: Diese Tracing-Funtionalität wird eventuell nicht nur in der Methode eineMethode() benötigt, sondern in vielen anderen auch, sodaß das Schreiben von Log-Nachrichten an vielen Stellen stattfinden muß. Bei einem notwendigen Austausch der Implementierung der Logging-Funktionalität, müssen alle diese Stellen entsprechend angepaßt werden.

```
public void eineMethode() {
    logger.trace("Betrete␣\"eineMethode\"");

    // Abarbeitung der Methode
    int i = 2 + 2;

    logger.trace("Verlasse␣\"eineMethode\"");
}
```

Quellcode 5.1: Tracing in Java ohne AOP

Mittels AOP[23] läßt sich das Tracing in einen Aspekt kapseln. Bevor in Codebeispiel 5.2 die Methode `quellMethode()` die Methode `eineMethode()` aufruft, wird der Tracing-Aspekt aktiv. Aufgrund eines im sogenannten „Pointcut“ definierten Patterns ergeben sich eine Menge von „Join Points“, unter anderem der Punkt vor Methode `eineMethode()`. Die Funktionen `before()` und `after()` im Tracing-Aspekt, die sogenannten „Advices“, bestimmen, daß bei Ein- und Austritt aus Methode `eineMethode()` Tracing-Nachrichten geschrieben werden. In diesem Falle ist es eine einfache Ausgabe an die Standardausgabe.

```
public aspect Tracing {
  // PointCut. Überwacht alle Methoden der
  // Klasse AOPDemo unabhängig von ihrer Signatur
    pointcut traceCall():
        call(* AOPDemo.*(..));

  // Before Advice
```

[23] eine Funktionalität, die in Java durch die AspectJ-Erweiterung geliefert wird

```
    before(): traceCall() {
        System.out.println("Betrete \"" + thisJoinPoint + "\"");
    }

  // After Advice
    after(): traceCall() {
        System.out.println("Verlasse \"" + thisJoinPoint + "\"");
    }
}

// ...

// Methode der Klasse AOPDemo
public void eineMethode() {
    // Join Point
    int i = 2 + 2;
}

// ...

public void quellMethode() {
    eineMethode();
}
```

Quellcode 5.2: Tracing in Java mit AOP Aspekt

An diversen Forschungseinrichtungen in Deutschland wird über dieses Thema geforscht [Berlin][Bonn][Darmstadt]. Es ist daher anzunehmen, daß die AOP in den nächsten Jahren, ähnlich wie die generische Programmierung derzeit, auf starke Akzeptanz stoßen wird. Sie sollte daher Bestandteil eines jeden Entwicklungs-Frameworks sein.

5.2.2. Inversion of Control

IoC ist eine Programmiertechnik, bei der ein Programm die Kontrolle über seine Ausführung abgibt und auf Aufrufe mitsamt Aufruf-Parametern an sich selbst wartet, um danach mit der Ausführung fortzufahren. Diese Technik, auch als Hollywood-Prinzip[24] bezeichnet, ist bei der Framework-Programmierung sehr wichtig [Fowler (2005)]. In diesem Zusammenhang soll jedoch nur eine spezielle Form der IoC von Interesse sein: Die Dependency Injection.

[24] „Dont call us, we call you“

Bei der DI-Programmiertechnik übergibt ein Objekt die Kontrolle über die Art wie es seine eigenen Attribute initialisiert an ein anderes Objekt ab. Ohne DI erzeugt das Objekt alle für seine Arbeit benötigten Objekte und speichert sie in seinen Attribut-Variablen. Dazu muß es allerdings genaue Kenntnis über die Implentierung der benötigten Objekte kennen, was die Kopplung erhöht. DI ist eine Technik, welche die Objekt-Kopplung verringert. Ein Pseudo-Code-Beispiel kann auf [Wikipedia (2007h)] nachgelesen werden.

5.2.3. Continuation Server

Continuation Server nutzen das Konzept der Continuations. Dadurch sind sie in der Lage die Ausführung von serverseitigem Programmcode zu jedem Zeitpunkt zu stoppen und zu einem späteren Zeitpunkt wieder aufzunehmen. Dies ermöglicht im Rahmen der Webprogrammierung durch HTTP-Anfrage-Antwort-Zyklen zeitlich unterbrochene Geschäftsprozeß-Abläufe trotzdem auf eine lineare Folge von Programmiersprachen-Statements abzubilden.

5.3. Java Software Stack

5.3.1. Aufbau

Die Komponenten einer Java-Webentwicklungsumgebung bauen auf der JavaEE-Spezifikation auf. Es handelt sich dabei um eine vom „Java Community Process“ entwickelte Spezifikation einer Softwarearchitektur für die Ausführung von in Java programmierten Anwendungen. In der Spezifikation werden als allgemein sinnvoll erachtete Dienste und Komponenten definiert, auf deren Basis modulare, mehrschichtige Anwendungen entwickelt werden können. Klare Schnittstellendefinitionen sorgen dafür, daß Implementierungen unterschiedlicher Hersteller zueinander kompatibel sind.

Die Laufzeitumgebung von JavaEE-Komponenten ist ein sogenannter JavaEE Application Server. Nach der aktuellen Spezifikation stellt er neben den üblichen Aufgaben einer Laufzeitumgebung den Zugriff auf Betriebssystem-Ressourcen[25] zu kapseln, auch Funktionalitäten zu Aspekten wie Sicherheit, Transaktionsmanagment, Kommunikation zwischen JavaEE-Komponenten und Installationsunterstützung zentral zur Verfügung. Logisch besteht ein JavaEE Server unter anderem aus einem EJB-Container als RTE für „Enterprise Java Beans“ und einem Web-Container als RTE für Servlets und „Java Server Pages“.

Enterprise Java Beans, Servlets und Java Server Pages sind Bestandteile der umfangreichen JavaEE-API, welche im Rahmen der Webprogrammierung wichtige Funktionalitäten bereit-

[25] Dateisystem, Netzwerk, ...

stellt. Tabelle 5.2 listet einen kleinen Auszug dieser Spezifikationen der API mit einer kurzen Beschreibung:

API	Beschreibung
EJB	Diese Spezifikation liefert eine standardisierte Möglichkeit immer wiederkehrende Aufgaben für Geschäftslogik-Komponenten wie Persistenz, transaktionale Integrität und Sicherheit in Form von EJBs zu lösen, sodaß Programmierer sich auf die eigentlichen fachbezogenen Aufgaben konzentrieren können. Sie beschreibt außerdem die Rolle des EJB-Containers und wie EJBs in diesen eingerichtet werden.
Servlet	Servlets sind nach der Spezifikation das Java-Gegenstück zu Technologien wie CGI. Ein Web-Container, der ein Bestandteil eines Webservers ist, kommuniziert mit einem Servlet. Dabei übergibt der Web-Container je nach URL einem bestimmten Servlet die Request-Daten und die Umgebungsvariablen der Laufzeitumgebung[26]. Dieses generiert daraufhin die Antwort. Servlets sind imstande den Zustand einer HTTP-Sitzung durch Cookies zu speichern.
JSP	Eine zu Servlets eng verwandte Technologie, mit der dynamisch HTML- und/oder XML-Code generiert werden. Statischer HTML- und XML-Inhalt wird mit zusätzlichen JSP-Actions versehen, welche durch Java-Code-Output ersetzt werden können, indem JSPs in Java Servlets kompiliert werden.

Tabelle 5.2.: Auszug aus der JavaEE API

Aufbauend auf der beschriebenen Spezifikation gibt es diverse Implementierungen für JavaEE-Server sowie separate Implementierungen für Web- und EJB-Container. Als Beispiele seien „JBoss“ oder „IBM Websphere“ für JavaEE-Server genannt, „Apache Tomcat“ für einen Web-Container sowie „Apache OpenEJB“ für einen EJB-Container [Wikipedia (2007o)]. Mit JBoss und Hibernate, einem OR-Mapper ließen sich bereits Webanwendungen schreiben. Das Konzept der EJB in der Version 1.0 und 2.0 stieß jedoch in der Java-Welt bis jetzt auf wenig Gegenliebe, da es eine zu schwergewichtige und komplexe Technologie ist [Wikipedia (2007l)]. Daher wurde als Gegenlösung im Juni 2003 Version 1.0 des sogenannten „Spring Frameworks“ basierend auf der Buch-Publikation „Expert one-on-one J2EE development without EJB“ [Johnson (2002)] von Rod Johnson herausgegeben. Das Anliegen Johnsons war es, die Zusammenarbeit von Anwendungen mit diversen Teilen der JavaEE-Plattform zu vereinfachen und zu vereinheitlichen. Das Framework sollte anerkannte, aber bis dahin wenig umgesetzte Praktiken der Softwareentwicklung aufgreifen und den JavaEE-Standards genügen. Zu diesen Praktiken

gehören AOP und IoC. Das Spring-Framwork verfügt über die in Tabelle 5.3 genannten kleineren Sub-Frameworks. Ein Java-Full-Stack ist somit durch die Kombination von JBoss als Laufzeitumgebung und Spring und Hibernate als Webanwendungs-Entwicklungs-Frameworks gegeben. Mit dieser Kombination läßt sich im IT-Geschäftsbereich nahezu jede Anforderung befriedigen. Tate vermerkt:

> „Java can talk to just about any enterprise system thats important to you."

Sub-Framework	Beschreibung
Inversion of Control Container	Ermöglicht die Konfiguration von Objekten und die Verwaltung des Objekt-Lebenszyklus
AOP-Framework	Spring hat ein eigenes AOP-Framework
Persistenz-Framework	Technologien wie JDBC, Hibernate und weitere werden unterstützt
Transaktions-Framework	Transaktion-Management für Java-Objekte
MVC	Framework für MVC
Framework für Entfernten Zugriff	Export und Import von Java-Objekten über Computernetzwerke durch RMI, CORBA und HTTP-basierten Protokollen inklusive SOAP
Authentifizierungs- und Autorisierungs-Framework	Breite Unterstützung durch gebräuchliche Industriestandards, Protokolle, Werkzeuge
Remote-Managment-Framework	Java Objekte können in puncto lokaler und remoter Nutzung transparent gemacht werden (JMX)
Testing framework	Für Klassen Unit- und Integrations-Tests geschrieben werden

Tabelle 5.3.: Spring Sub-Frameworks

5.3.2. Fazit

Bei der Recherche über den Aufbau eines Java-Webentwicklungs-Frameworks fiel dem Autor vor allem die Vielfalt von zur Verfügung stehenden Frameworks auf. Der Umfang an Dokumentation ist gewaltig. Allein für die Kombination von JBoss, Spring und Hibernate müssen gut 1700 Pdf-Seiten an Referenzdokumentation konsumiert werden.[27] Tate geht bei seiner Empfehlung über zu benutzende Frameworks sogar noch einen Schritt weiter. Folgende Frameworks gehören für ihn zu einem Full Stack: Ant, um Webanwendungen zu deployen, Tapestry als MVC-Framework, Hibernate als ORM und Spring, um die Anwendung testbar zu machen

[27] nachzulesen auf den offiziellen Internet-Sites

und Funktionalitäten wie Sicherheit und verteilte Transaktionen für POJOs verfügbar zu machen. Über die dazugehörige Lernkurve schreibt er:

> „My clients that move to Java from another language just shudder when they see my recommendation of five weeks of education, which lets them cover only the fundamentals“

Angesichts der Tatsache, daß nach Tate mehr als 50% der Anforderungen in der heutigen, globalisierten Geschäftswelt die Verwaltung einer Datenbank mittels eines Web-Frontends ist, erscheint dieser Overhead nicht akzeptabel. Er merkt an, daß man im Java-Umfeld hart arbeiten müsse, um diese Anforderung zu erfüllen, weil man leicht im Überangebot der gebotenen Features, die zudem oft nur marginale und nichtzentrale Probleme abdecken, ertrinken könne. Die Notwendigkeit des massiven Einsatzes von XML zur Konfiguration der Softwarekomponenten, die in Javas Unvermögen zum Ausdrücken strukturierter Daten liege, wirke sich auch produktivitätsmindernd aus.

Die Nutzung von Techniken wie AOP und DI sind als positiv hervorzuheben, jedoch fällt bei der noch folgenden Untersuchung von Frameworks dynamischer Sprachen auf, daß Anforderungen, die diese Paradigmen erfordern, durch wesentlich leichtgewichtigere Techniken auf Sprachebene implementiert werden können.

Desweiteren vermißt man bei Java neuartige Konzepte wie Continuation Server gänzlich.

Wie in Abschnitt 5.3.1 gesehen, kann Java mit der Vielzahl an Frameworks nahezu sämtliche Anforderungen in der IT-Geschäftswelt befriedigen. Als abschließende Bemerkung über die Rolle Javas sei noch einmal auf ein Zitat von Tate verwiesen:

> „They [Autor: gemeint ist Java und zugehörige Framworks] target very complex problems at the expense of the easy problems that most Java developers need to solve. [...] Java has become strictly a language for hard-core enterprise development of large-scale problems.“

5.4. Ruby on Rails

Ruby on Rails, abgekürzt Rails, ist ein von David Heinemeier Hansson in Ruby geschriebenes und quelloffenes Web-Framework, das im Juli 2004 zum ersten Mal der Öffentlichkeit vorgestellt wurde [Wikipedia (2007s)].

5.4.1. Philosophie

Rails basiert auf zwei Gedanken, auf denen sich der gesamte Aufbau und die Nutzung des Frameworks gründen:

Don't repeat yourself: Nichts soll irgendwo zweimal codiert werden. Inkonsistenzen und die Notwendigkeit bei Änderungen mehrere Stellen zu ändern, sollen so vermieden werden. Dank Active Record ist es daher ausreichend die Spalten einer Tabelle nur in der Datenbank festzulegen, da Rails zur Laufzeit diese Informationen aus der Datenbank liest sowie Getter- und Setter-Methoden erstellt. Eine zweite Konfigurationsdatei kann so entfallen.

Convention over configuration: CoC bedeutet, daß Rails sinnvolle Standardwerte erwartet. Beispiele hierfür sind:

- Der Primärschlüssel einer Datenbanktabelle heißt „ID“ und ist vom Typ „Integer“
- Ein Model mit dem Namen `Customer` ist in der Datei `#{RAILS_ROOT}/app/models/customer` gespeichert. Die zugehörige Datenbanktabelle heißt `customers`.
- Ist das Model `Customer` mit dem Model `Contract` über eine 1:N-Beziehung miteinander verknüpft, so erwartet Rails, daß in der Tabelle `contracts` ein Fremdschlüssel mit dem Namen `customer_id` vorhanden ist.

Auf Model-Ebene brauchen dann nur noch wenige Angaben zu den Assoziationen gemacht werden[28] und man kann dann dank Active Record alle Contracts eines Customer-Objekts so aufrufen: `customer1.contracts`. Man muß sich nicht an diese Standardwerte halten, da sie leicht umkonfiguriert werden können.

5.4.2. Scaffolding

Der Rails-Scaffolding-Mechansimus unterstützt einen in vielfältiger Weise beim Beginn der Entwicklung einer Webanwendung. Diverse Scripte, die in einer Konsole ausgeführt werden, legen die initiale Verzeichnisstruktur, Controller mit dazugehörigen CRUD- und REST-Methoden, Models, Views, Datenbankmigrationen und Testsuiten eines Projektes an.

Beispiele:

[28] dazu mehr in Absatz 5.4.3.1

Der Scriptaufruf `rails gista` legt ein Projekt namens gista im aktuellen Verzeichnis an. Dies erzeugt immer die gleiche Startstruktur. Unter anderem werden die in Tabelle 5.4 angeführten Unterverzeichnisse angelegt.

Unterverzeichniss	Beschreibung
app	In diesem Verzeichnis befindet sich der Anwendungscode. Für jede MVC-Komponente gibt es ein Unterverzeichnis.
config	Dieses Verzeichnis enthält Rubycode-Konfigurationsdateien. Zum Beispiel die Verbindungsparamter zur Datenbank. Durch die CoC-Philosophie bleibt dieses Verzeichnis klein.
script	Dort befinden sich die oben angesprochenen Skripte sowie Startskripte, um einen Webserver zu starten.

Tabelle 5.4.: Einige Rails-Projekt-Unterverzeichnisse

Der Scriptaufruf `ruby script/server` startet den Webserver. Die Aufrufparameter können einfach im Scriptverzeichnis per Ruby-Code konfiguriert werden (siehe Codebeispiel 5.3) oder auch einfach als Aufrufparameter übergeben werden.

```
OPTIONS =
{
  :port        => 3000,
  :ip          => "0.0.0.0",
  :environment => "development",
  :server_root => File.expand_path(File.dirname(__FILE__) + "/../public/"),
  :server_type => WEBrick::SinpleServer
}
```

Quellcode 5.3: Ruby Options Hash

In `config/database.yml` (siehe Codebeispiel 5.4) wird Rails mitgeteilt, wo die Datenbank liegt und von welchem Typ sie ist.[29]

```
development:
  adapter: mysql
  databse: trails
  host: localhost
  username: root
```

[29] Active Record kann mit folgenden Datenbanken kommunizieren: DB2, Informix, Firebird, MySQL, Openbase, Oracle, PostgreSQL, SQLite, SQL Server, Sybase

```
  password: password
```

Quellcode 5.4: Ruby YAML DB Config File

Der Skriptaufruf `ruby script/generate model route titel:String` generiert ein Route-Model, einen Test-, Fixture- sowie Migration-Stub in den passenden Verzeichnissen. Außerdem wird auch gleich String-Attribut namens „titel“ angelegt. Dabei entsteht der Model-Code in Codebeispiel 5.5. Dieser ist erstaunlich klein. Dies liegt an der von Rails verfolgten CoC-Philosophie. Wenn man sich an die Konvention hält und eine Datenbanktabelle namens `routes` anlegt, dann erstellt AR zur Laufzeit passende Zugriffsmethoden für jedes Route-Objekt. Entsprechend gibt es unter anderem Generatorskripts für Controller, Migrationen, Plugins.

```
class Route < ActiveRecord::Base
end
```

Quellcode 5.5: generierte Ruby Klasse

5.4.3. Aufbau

Ruby on Rails bietet alle Sub-Frameworks „aus einer Hand“, um damit komfortabel Webanwendungen zu entwickeln (siehe Tabelle 5.5).

Im folgenden soll auf die beiden Frameworks Active Record und Action Pack näher eingegangen werden.

5.4.3.1. Active Record

Active Record implementiert das „Active Record“-Entwurfsmuster von Martin Fowler [Fowler u. a. (2003)]. Es handelt sich um einen Wrapper um eine Datenbanktabelle, der eine elegante Brücke zwischen der fachlichen Objektwelt und der technischen, relationen Datenbank-Welt schlägt. Dabei fungiert AR als DSL. Das heißt, es kann sehr abstrakt auf Objekte und auf die mit ihnen assozierten Objekte zugegriffen werden (Association Managment). Folgende Punkte gehören unter anderem zu den innovativen Features:

- Automatic properties
- Assciation management
- Composition

Sub-Framework	Beschreibung
Active Record	Objektabstraktionsschicht basierend auf dem objekt-relationalen Muster oder Model (das „M“ aus MVC) von Martin Fowler
Action Pack	Request-Behandlung und Response-Ausgabe. Die Anfragen werden durch eine öffentliche Methode des Controllers (Action Controller, das „C“ aus MVC) behandelt. Die Ausgabe wird mittels eines Templates (Action View, das „V“ aus MVC) vorgenommen.
Active Support	Ruby-Erweiterungen von Rails
Action Mailer	E-Mail-Versand und -Empfang
Action Web Service	Web-Service-Programmierung, Unterstützung für SOAP, XML-RPC und REST.

Tabelle 5.5.: Ruby on Rails Sub-Frameworks

- Inheritance

Automatic properties Rubys Reflection und CoC sorgen dafür, daß über ein Model-Objekt durch zur Laufzeit erstellte Akzessoren auf korrespondierende Datenbankattribute zugegriffen werden kann. Der Model-Code wird von technischen ORM-Aspekten befreit.[30]

Association Management Durch hinzufügen spezieller Methoden in den Model-Code können Assoziationen automatisch hergestellt werden ohne das Fremdschlüsselbeziehungen auf Datenbankebene definiert werden müssen.[31]

Beispiel: Gegeben sei der GISTA-Datenmodell-Entwurf in Abbildung 5.1. Die Assoziationen werden im Model-Code gemäß Code-Beispiel 5.6 hergestellt. Durch diese Verknüpfungen kann nun beispielsweise in einem Controller auf die Routen eines Nutzers vermöge `user1.routes` zugegriffen werden. Oder vermöge `user1.routes << aRoute` eine Route hinzugefügt werden.[32]

```
class User < ActiveRecord::Base
  has_many :routes
  has_many :points
```

[30] siehe Codebeispiel 5.5

[31] was aber auf Wunsch ohne weiteres möglich ist

[32] für weitere Möglichkeiten siehe Modul ActiveRecord::Associations auf [Opensource (e)]

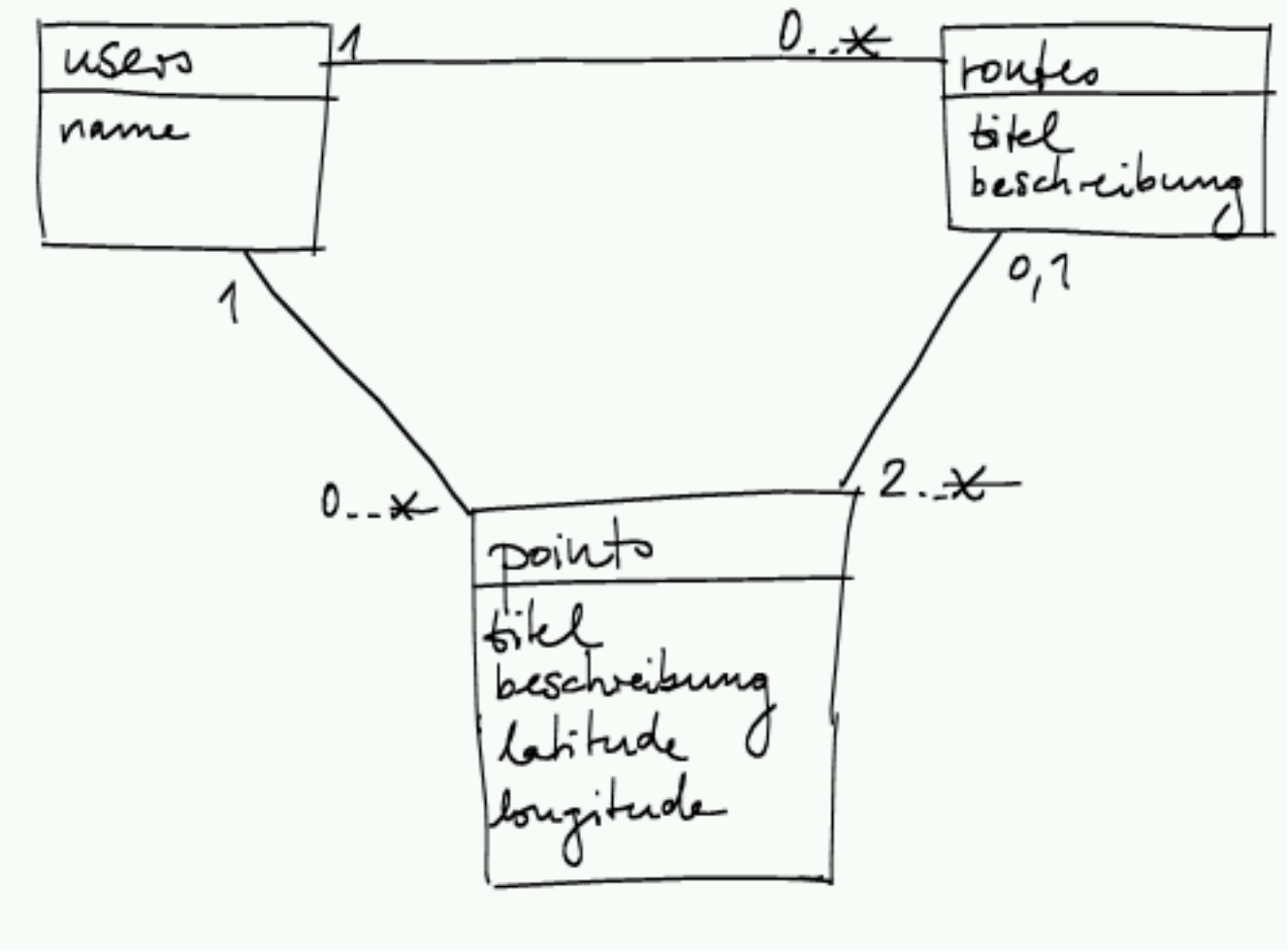

Abbildung 5.1.: Datenmodell-Entwurf des GISTA

```
end

# ...

class Route < ActiveRecord::Base
 belongs_to :user
 has_many :points
end

# ...

class Point < ActiveRecord::Base
 belongs_to :routes
 belongs_to :user
end
```

Quellcode 5.6: Active Record Assoziationen

Composition Dabei handelt es sich um eine Art Tabellen-View auf Model-Ebene. Das `Location` Model verfügt über ein Attribut `name`, welches das Attribut `street` aus dem `Street` Model ist.

```
class Location < ActiveRecord::Base
 composed_of :street, :class_name => "Street",
       :mapping => %w(street name)
end
```

Quellcode 5.7: Active Record Compositions

Inheritance Vererbung unter Models funktioniert, indem man einfach auch alle Unterklassen in die Oberklassendatei schreibt.

```
class Product < ActiveRecord::Base
end
class Bike < Product
end
```

Quellcode 5.8: Active Record Inheritance

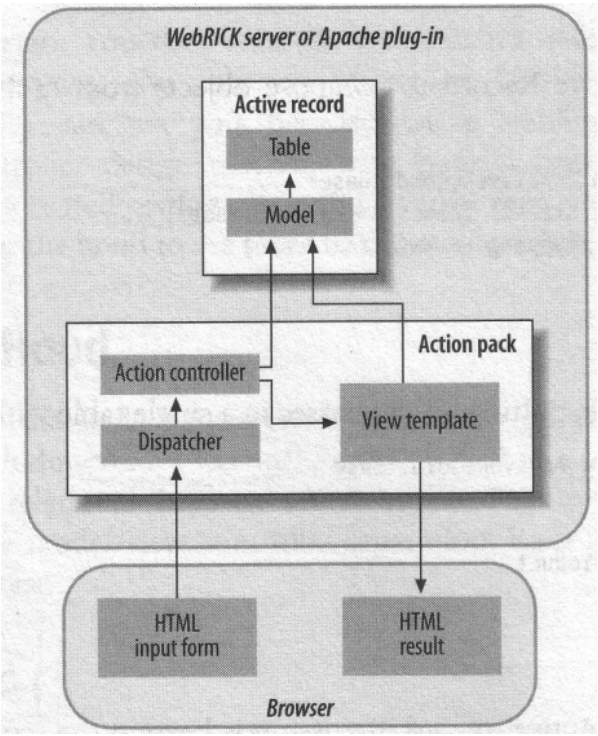

Abbildung 5.2.: Ruby on Rails Architektur

5.4.3.2. Action Pack

Action Pack nimmt Anfragen entgegen und verarbeitet sie gemäß Ablaufdiagramm in Abbildung 5.2. Eine HTML-Anfrage wird von einem Dispatcher entgegengenommen, der diese dann an den entsprechenden Controller weiterleitet. Der Controller führt die passende Aktion aus, indem er eventuell AR benutzt, um auf Datensätze der Datenbank zuzugreifen. Außerdem startet er die Template-Engine und stellt dem View seine ermittelten Datensätze zur Verfügung, die in ihm an passenden Stellen eingebaut werden.[33] Der genrierte HTML-Code wird an den Browser geschickt.

Action Pack stellt folgende Fähigkeiten zur Verfügung [Opensource (e)]:

- Controller-Aktionen werden im Controller als Ganzes gespeichert und nicht als separate Kommando-Objekte wie im MVC-Java-Framework Struts. Dadurch können sie auf gemeinsame Helper-Methoden zurückgreifen.
- Ruby wird als Template-Sprache benutzt.[34] Es wird keine separate Sprache benötigt.

[33] Das View Template kann auch selbst auf AR zugreifen, was aber vermieden werden sollte, da Geschäftslogik besser im Controller aufgehoben ist.

[34] ERB: Embedded Ruby

- Durch eine DSL in Form von Buildern können leicht XML-Ausgaben erzeugt werden. Zum Beispiel für RSS.
- In Controller-Aktionen können Filter gesetzt werden. Vor und nach der Ausführung Methoden können beliebige andere Methoden ausgeführt werden.[35]
- Formulare, das Datum, Links und Text können durch Hilfs-Methoden im View generiert werden.
- Ein Website-Layout kann gekapselt und wiederverwendet werden (entspricht einer einfachen Version der „Tiles“ in Struts)
- Durch einen Routing-Mechanismus können URLs beliebig gewählt werden. Sie werden daraufhin dem richtigen Controller und der passenden Aktion zugeordnet.
- Javascript und Ajax sind nahtlos integriert.
- automatisches Paginieren: Datenbanksätze können so auf mehrere Seiten verteilt werden, auf die über Seitenzahl-Links zugegriffen werden kann.
- Unit-Testing von Controller- und View-Ausgaben
- Automatisches Benchmarking und Logging.
- Caching auf drei Ebenen (Seite, Aktion, Fragment)[36]
- Fürs Debugging werden für lokale Nutzer Fehler-Nachrichten, Stack-Trace, Anfrage-Paramter, Session-Inhalte und die zum Teil erstelle Antwort im Browser ausgegeben.
- Scaffolding auf Controller-Ebene. Für den CRUD-Zugriff auf Models werden alle Methoden automatisch erstellt.

Vergleicht man dieses Angebot an Features mit den in Tabelle 5.1 auf Seite 43 geforderten, stellt man fest, daß nur noch Transaktionsmanagment, die Valiedierung von Eingaben, Internationalisierung und Lokalisierung, sowie AOP und IoC bzw. DI fehlen. Die ersten beiden Punkte sind Bestandteil von Active Record. Der dritte Punkt kann leicht durch ein Plugin nachgerüstet werden. Auf die anderen beiden Punkte wird in den folgenden Abschnitten eingegangen.

5.4.4. AOP in Rails

Das Konzept wird in Abschnitt 5.2.1 erläutert. Java besitzt mit der AspectJ-Erweiterung oder dem Spring-Framework AOP-Unterstützung. In Rails gibt es kein Pendant dafür. Allerdings

[35] unter anderem sinnvoll für Authentifizierung

[36] Fragment bezieht sich auf das Zwischenspeichern von Ausgaben, die noch kleiner als eine Aktion sind. Siehe dazu ActionController::Caching::Fragments auf [Opensource (e)]

kann dieses Programmierparadigma mit drei Konzepten der Sprache Ruby implementiert werden [Tate (2005)]:

- Aliasing[37]
- Mixins[38]
- Ruby hooks __before, __after, __wrap[39]

5.4.5. DI in Rails

Das Konzept wird in Abschnitt 5.2.2 erläutert. Zur die Wichtigkeit dieses Paradigmas in „Ruby on Rails“ sei Jim Weirich aus [Tate (2005)] zitiert:

> „DI is a tool to help you build decoupled programs [...] You start using DI when the pain of coupling outweights the cost of the additional complexity. In some languages, this pain comes fairly quickly. But in Ruby, the pain is mitigated by the natural tendency for looser coupling in a dynamically typed language, so the benefits of dependency injection occur much later in the cost curve. [...] And finally, when your application/framework does grow to the point that dependency injection becomes interesting, you will find that a simple (i.e., less than 30 lines of code) dependency injection library will fill most of your needs.“

5.4.6. Fazit

Die Arbeit mit Ruby on Rails gestaltet sich aus der Sicht des Autors als sehr komfortabel. Alle Aufgaben eines Webframeworks (siehe Tabelle 5.1) werden durch Bordmittel des Frameworks und der Sprache Ruby auf eine einfache und verständliche Weise gelöst. Es existiert eine einheitliche, übersichtliche API für das Framework und die Sprache Ruby, die elegant von allen wichtigen Aufgaben des Frameworks abstrahiert [Opensource (e)][Opensource (d)][Opensource (f)]. Die ausgezeichneten Metaprogrammierungs-Fähigkeiten von Ruby ermöglichen die Umsetzung von CoC im AR-Framework, sodaß Zugriffsmethoden zur Laufzeit erstellt werden können. Dadurch entfällt die Notwendigkeit fachlich irrelevanten Zugriffscode zu pflegen und zu verstehen. Über CoC schreibt Justin Gehtland in [Tate (2005)]:

> „[...] the dedication to smart defaults is the primary benefit to Ruby on Rails. [...] you can create an application and ignore 90% of what would be surfaced in configuration files in another framework“

[37] Siehe Abschnitt 3.2.2.11 für Definition und Code-Beispiel 3.6 auf Seite 15
[38] Siehe Abschnitt 3.1.1
[39] nach Tates Aussage in Ruby Version 2.0 verfügbar

Über die Metaprogrammierungs-Fähigkeiten von Ruby in Rails schreibt Bruce Tate:

> „Rails is one of the first commercially successful demonstrations of metaprogramming in Ruby, in combination with meaningful defaults.“

Tabelle 5.6 zeigt die Ergebnisse einer Produktivitätsmessung für die Umsetzung eines von Tates Softwareprojekten mit Java, Spring, Hibernate einerseits und Ruby on Rails andererseits [Tate (2005)].

Metric	Java/Spring/Hibernate	Ruby on Rails
Time to market	4 months, approximately 20 hours/week	4 nights (5 hours/night)
Lines of Code	3.293	1.164
Lines of configuration	1.161	113
Number of classes and methods	62/649	55/126

Tabelle 5.6.: Produktivitätsvergleich Java/Spring/Hibernate und Ruby on Rails

Weiterhin unterstützen sinnvolle Generator-Skripte Scaffolding und befreien einen von immer wiederkehrenden lästigen Aufgaben.

5.5. Seaside

Seaside ist ein Open Source Web-Entwicklungsframework, um Web-Anwendungen in Smalltalk zu schreiben [Wikipedia (2007t)][Opensource (g)].

5.5.1. Aufbau

Versucht man das Seaside Framework mit Ruby on Rails zu vergleichen, so trifft der Vergleich in Tabelle 5.7 am ehesten zu. Für alle wichtigen Aufgaben der Webentwicklung gibt es adäquate Lösungen für Seaside. Das Framwork kann als Modul zusammen mit allen anderen Komponenten innerhalb der Smalltalk-Entwicklungsumgebung Squeak oder VisualWorks geladen werden.

Seaside kommt zusammen mit einem Continuation Server, der neuartige Ansätze bei der Webprogrammierung ermöglicht.

Rails Komponenten	Seaside-Komponenten
Active Record	Glorp
Action Pack	Seaside
Active Support	Vorhandener Squeak Code
Action Mailer	Vorhandener Squeak Code
Action Web Service	SoapCore

Tabelle 5.7.: Smalltalk Komponenten im Vergleich zu Ruby on Rails

5.5.2. Integration von Continuations

Die Folgenden Erklärungen stammen aus einem Artikel von Ducasse, Lienhard und Renggli [Ducasse u. a. (2004)]. Alle bisher vorgestellen Webentwicklungs-Ansätze bieten keine durchgehend konsistente Möglichkeit einen Geschäftsprozeß, der sich auf Anwenderseite durch eine bestimmte Folge Web-Seiten zeigt[40], auf einem abstrakten Niveau zu definieren. In Seaside läßt sich ein Geschäftsprozeß für einen Internet-Shop, wie er in Abbildung 5.3 dargestellt ist, durch serverseitigen Code, wie in Beispiel 5.9 abbilden.

```
StoreTask >> go
 | cart shipping billing creditCard |
 cart := StoreCart new.
 [ self fillCart: cart
  self confirmContentsOfCart: cart ] whileFalse.
 shipping := self getShippingAddress.
 billing := (self useAsBillingAddress: shipping)
  ifFalse: [ self getBillingAddress ]
  ifTrue: [ shipping ].
 creditCard := self getPaymentInfo.
 self ship: cart to: shipping billTo: billing payWith: creditCard.
 self displayConfirmation.
```

Quellcode 5.9: Geschäftsprozeß in Seaside

Ein wichtiger Punkt bei diesem Ansatz ist das Zusammenfügen von Präsentations- und Controller-Code in sogeannte Seaside-Komponenten, die serverseitg als Smalltalk-Objekte implementiert sind. Sie verfügen daher über die Fähigkeit ihren Zustand zu verwalten und sich

[40] in einem Online-Buchladen zum Beispiel eine definierte Folge von Seiten, welche zunächst die Artikelliste mit Einkaufswagen, eine Bestell-Bestätigungs-Seite, die Abfrage persönlicher Daten und dann die Abfrage von Kreditkartendaten zeigt

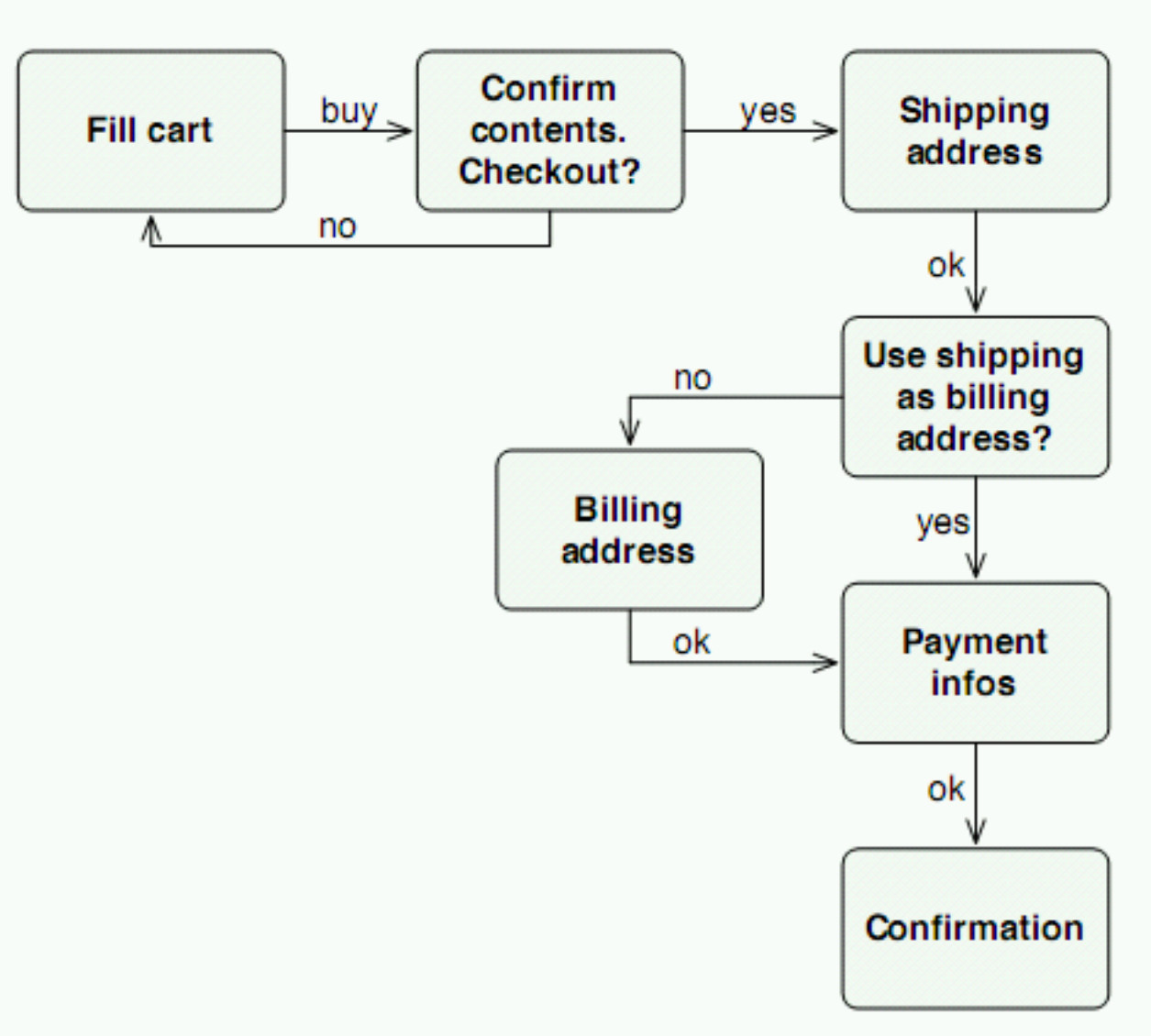

Abbildung 5.3.: Geschäftsprozeß für einen Bestellvorgang in einem Internetshop

selbst zu rendern, also eine HTML- oder XML-Darstellung von sich zu liefern. Der Zustand zwischen Client und Server wird intern von Seaside über kryptische URL-Parameter über die Anfrage-Antwort-Zyklen gerettet. Dem Problem der Verwendung des „Zurück“-Knopfes, der zu einem inkonsistenten Zustand zwischen dem Model auf Serverseite einerseits und der Darstellung im Browser andererseits führt, wird in Seaside mit einem Backtracking-Mechanismus begegnet: Nach jeder Client-Antwort wird ein Schnappschuß der registrierten Objekte durch Speicherung in einen Cache vorgenommen. Durch Continuations wird zu jedem Schnappschuß auch der entsprechende Ausführungskontext gespeichert. Auch Transaktionen, also ein Geschäftsprozeß, der nach dessen Komplettierung nicht wieder in Teilen wiederholt werden darf[41], können leicht implementiert werden.

5.5.3. Die Entwicklungsumgebung

Smalltalk-IDEs wie Squeak oder VisualWorks bieten gute Unterstützung fürs Debuggen an. Code kann zur Programmlaufzeit geändert werden ohne der Notwendigkeit ihn zu Rekompilieren oder einen Server neuzustarten.[42]. Diese Eigenschaft wird wird von Seaside genutzt, um browserseitiges Debuggen zu ermöglichen. Bei Rails wird im Falle eines Laufzeitfehlers ein Fehlername, der Inhalt der Anfrage, die Antwort in Teilen und der Call-Stack auf einer statischen HTML-Seite präsentiert. Weitere Informationen müssen aus Logdateien geholt werden. Seaside geht weit darüber hinaus: Über eine Toolbar am unteren Rand des Browserfensters, welche Bestandteil der HTML-Ausgabe ist und nach Belieben ein- und ausgeschaltet werden kann, lassen sich neben anderen Informationen Debugging-Elemente per Klick hervorholen. Jede Komponente[43] wird auf der Seite mit ihrem Objektnamen gekennzeichnet. Über drei Knöpfe dahinter läßt sich ein „System Browser“, ein Inspector oder ein „Library Browser“ aufrufen. Der System Browser zeigt das Objekt in der Klassenhierarchie mit Methodenprotokoll und Implementierung und der Möglichkeit diese zu ändern. Der Inspector liefert eine Zusammenfassug des aktuellen Objekt-Zustands. Und mithilfe des Library-Browsers können zur Laufzeit assoziierte CSS-Dateien editiert werden.

5.5.4. Fazit

Wie bisher ersichtlich ist Smalltalk Ruby sehr ähnlich. Die gute Integration des Continuations Konzeptes ermöglcht es Seaside, Geschäftsprozesse bei der Entwicklung von Webanwendungen auf einem abstrakten Niveau zu implementieren. Der Ansatz in Geschäftsprozessen

[41] Zum Beispiel ein Internet-Shop-Bestellvorgang. Mögliches Sicherheitsrisiko: Bestellen von Ware und Zahlen des Preises per Kreditkarte. Danach wieder zurück gehen und weitere Ware hinzufügen

[42] bei Rails auch möglich

[43] z.B. Artikelliste, Einkaufswageninhalt usw.

zu denken erscheint jedoch zunächst recht ungewohnt. Der Erfinder von Seaside, Avi Bryant, schreibt in seinem Blog auf der Concom Smalltalk Website [Bryant (2005)]:

> „The dominant paradigms and philosophies of web development – CGI, Servlets, Server Pages, REST– provide only a thin wrapper around the low-level details of HTTP, and encourage you to use the rough stones of the transport protocol as the direct building blocks of your application. Web developers by and large reject any further abstraction in the way that assembly hackers once rejected structured programming“

Aus der Sicht des Autors bietet Seasides einzigartiges Konzept zukünftigen Anforderungen durch immer gesteigerte Komplexität der Webapplikationen gerecht zu werden. Die Präsentations- und Controllerschicht sind in Komponenten gekapselt, die als Smalltalkobjekte auf dem Server leben. Dies ermöglicht eine verbesserte Wiederverwendung und Wartung von Code. Als negativ ist die Tatsache zu erachten, daß alle Komponenten-Objekte und Continuations während der gesamten Nutzersitzung im Serverspeicher gehalten werden müssen. Continuations könnten in einer Datenbank aufbewahrt werden, jedoch impliziert dies die Notwendigkeit diese zu serialisieren, was nicht trivial ist. Das letzte Release (2.8) konnte jedoch seinen Speicherhunger von vorher typischerweise 200kb auf 50kb senken [Wikipedia (2007t)]. In Seaside wurden bislang wenig Projekte realisiert, sodaß keine Erfahrungswerte für die Produktivität des Frameworks existieren.

6. Diskussion

6.1. Ergebnisse

6.1.1. Sprach-Ebene

Ruby- und Smalltalk-Code sind schlank und ausdrucksstark. Beide Sprachen bieten elegante Implementierungen für Konzepte wie Closures und dynamische Typisierung. Sie sind durchgehend objektorientiert, was Verständlichkeit und Erlernbarkeit fördert. Primitive in Java verletzen das Konzept der Objektorientierung. Dies zwingt zu einem Gebrauch von Wrappern, was zu Verwirrung und Uneinheitlichkeit führt. Ein kurzer „Code Compile Cycle“ bei den dynamischen Sprachen Ruby und Smalltalk ermöglicht ein schnelles Feedback beim Entwicklen und unterstützt damit Prototyping im Rahmen agiler Software-Entwicklungsmethoden. Reflection wird in allen Sprachen unterstützt. Ruby bietet eine besonders natürliche und elegante Implementierung, was diese Sprache als Framework-Programmiersprache prädestiniert. Desweiteren bietet nur Ruby das Aliasing-Konzept sowie Mixins als Implementierung für Mehrfachvererbung an. Mit beiden Mitteln kann man AOP realisieren. In Java benötigt man für AOP ein eigenes Framework. Das Continuations-Konzept wird von Ruby und Smalltalk unterstützt. Von Java gar nicht. Das DBC-Konzept wird nur von Ruby duch eine Bibliothek unterstützt. Java-Code wirkt oft aufgebläht und unübersichtlich. Dies hat mehrere Gründe: Zum einen ist da die Art der Typisierung zu nennen. In Ruby und Smalltalk ist sie dynamisch, in Java statisch. Die Folge: Im Gegensatz zu Smalltalk und Ruby muß in Java jede Variable typisiert werden. Dies führt zu mehr Codezeilen, die alle gewartet und verstanden werden müssen und zwingt zu verführter Detailarbeit. Desweiteren ist Javas Typsystem inkonsequent umgesetzt. Generics liefern nicht die gewünschte Typsicherheit. Ein Vorteil der statischen Typisierung ist die Fähigkeit der IDE Auto-Komplettierung anzubieten. Java läuft auf einer ausgereiften VM. Sie bietet auf Bytcode-Ebene Sicherheit in puncto Pufferüberlaufe und bei der Ausführung remoten Codes. Andere Sprachen könnten vom Reichtum von Javas Klassenbibliotheken profitieren, wenn sie auf der JVM laufen würden. Ruby und Smalltalk sind sich tendenziell sehr ähnlich.

6.1.2. Framework-Ebene

Rails und Seaside profitieren von den guten Reflection-Eigenschaften der Sprachen Ruby und Smalltalk. Ruby on Rails bietet Anfängern einen relativ leichten Einstieg, da alle wichtigen Aufgaben eines Web-Entwicklungsframeworks in einer klaren, einheitlichen Architektur vereint werden. Viele schwergewichtige Frameworks für AOP und DI können aufgrund von Ruby-Spracheigenschaften entfallen. Alles in allem stellt Tate eine signifikante Produktivitäts-Steigerung im Vergleich zu einem Java-Full-Stack fest. Ein solcher läßt sich aus einer Vielzahl verschiedener Frameworks zusammenstellen, die sich oft in den Aufgaben überschneiden. Der Umgang mit jedem Framework und ihr Zusammenspiel muß verstanden werden, was einen großen Einarbeitungsaufwand bedeutet. Dafür bieten diese Frameworks für alle erdenklichen Anforderungen Lösungern an. Seaside integriert das Continuations-Konzept in einem Continuation Server. Die Ausführung von Programm-Code kann daher zu jedem Zeitpunkt gestoppt und wieder aufgenommen werden. Ein Geschäftsprozeß-Ablauf, der normalerweise durch eine Folge von HTTP-Anfrage-Antwort-Zyklen unterbrochen wird, kann daher durch ein einziges Stück Code repräsentiert werden, in dem die Berücksichtigung der HTTP-Zustandslosigkeit nicht mehr notwendig ist. View- und Controller-Code wird in Seaside-Objekten gekapselt, die ihren Zustand kennen und sich selbst rendern können.

6.2. Offene Fragen

Sprachebene: Interessant wäre eine weitere Untersuchung von Konzepten wie DBC und wie sich dessen Umsetzung auf die Produktivität einer Sprache auswirkt. Desweiteren wäre es wünschenswert präzise Metriken zur Beschreibung der Eleganz einer Sprache zu formulieren, damit sich die Ergenisse quantifizieren lassen.

Frameworkebene: Eine empirische Untersuchung über die tatsächliche Produktivität der betrachteten Frameworks wäre wünschenswert. Verschiedenen Entwicklerteams könnte jeweils eines der besprochenen Frameworks zugeordnet werden. Alle müssten dann den gleichen Entwurf einer Anwendung[44] implementieren. Mit vorher definierten Metrikmaßen wäre dann ein detailierter Vergleich hinschtlich Projektdauer, LOC, Fehlerhäufigkeit und weiteren Maßen möglich.

[44] zum Beispiel das GISTA

6.3. Ausblick

Der TIOBE-Index kann für jede Programmiersprache als Maß ihre Popularität genutzt werden. Er ergibt sich aus der Häufigkeit der Treffer für das Suchwort `<Sprache> + programming` unter Einbezug vieler gängiger Suchmaschinen [TIOBE]. Für November 2007 steht der Index in Abbildung [TIOBE]. Java steht auf Platz eins, Ruby auf Platz neun, Smalltalk auf Platz 36. Bei der Frage, ob sich eine Technologie etabliert, muß man auch schauen, ob sich eine florierende Community um sie gebildet hat. Technisch gutes setzt sich nicht immer durch. Vermarktungs- und Politik-Aspekte, die Trägheit großer Firmen und das Vorhandensein von Legacy-Systemen spielen oft eine größere Rolle. Denn was nützt es, wenn ein Konzept zwar großartig ist, aber es keine Entwickler gibt, die damit arbeiten wollen. Abbildung 6.2 zeigt die Häufigkeit der Suchanfragen für die Suchstrings: „java sun|hibernate|spring“, „ruby on rails“, „smalltalk seaside“. Für Rails interessiert man sich seit Anfang 2005. Java und seine Frameworks scheint seit diesem Zeitpunkt im Abwind zu liegen. Das Interesse für Seaside ist im Vergleich so niedrig, daß es praktisch als Null erachtet werden kann.

6.4. Konklusion

Viele Webapplikationen lassen sich mit Rails mindestens ebenso gut wie mit JavaEE entwickeln. Vor allem, wenn es nur darum geht eine Datenbank über ein Web-Frontend zu verwalten. Die entstehenden Anwendungen verfügen über eine saubere Architektur. Von Fall zu Fall muß jedoch entschieden werden, was das richtige Framework für das anstehende Projekt ist. Denn viele starten nicht von der grünen Wiese. Man betrachte einfach Systemteile, die in Java programmiert wurden. Der Legacy-Support ist in Java besser. Seaside verfügt über innovative Ansätze im Bereich der abstrakten Modellierung von Workflows innherhalb einer Web-Applikation. Das Framework arbeitet nur mit Objekten. Wiederverwendung wird sehr gefördert. Die Community-Unterstützung ist jedoch praktisch Null, sodaß man zur Zeit wenig Fachkräfte findet, die sich mit Smalltalk oder Seaside auskennen.

Position Nov 2007	Position Nov 2006	Delta in Position	Programming Language	Ratings Nov 2007	Delta Nov 2006	Status
1	1	=	Java	20.542%	+0.14%	A
2	2	=	C	13.969%	-3.23%	A
3	4	↑	(Visual) Basic	10.228%	+0.76%	A
4	3	↓	C++	8.750%	-2.30%	A
5	5	=	PHP	8.687%	-0.52%	A
6	6	=	Perl	4.738%	-1.49%	A
7	7	=	Python	4.227%	+0.59%	A
8	8	=	C#	3.917%	+0.89%	A
9	12	↑↑↑	Ruby	3.084%	+1.37%	A
10	9	↓	JavaScript	2.928%	+0.62%	A
11	10	↓	Delphi	2.456%	+0.20%	A
12	14	↑↑	D	1.704%	+1.02%	A
13	13	=	PL/SQL	1.179%	-0.04%	A
14	11	↓↓↓	SAS	1.129%	-1.08%	A
15	16	↑	Lisp/Scheme	0.754%	+0.17%	A--
16	48	↑↑↑↑↑↑↑↑↑	Lua	0.746%	+0.66%	A--
17	17	=	COBOL	0.708%	+0.14%	A--
18	15	↓↓↓	ABAP	0.647%	+0.01%	A--
19	19	=	Pascal	0.639%	+0.12%	B
20	18	↓↓	Ada	0.613%	+0.07%	B

Abbildung 6.1.: TIOBE Index

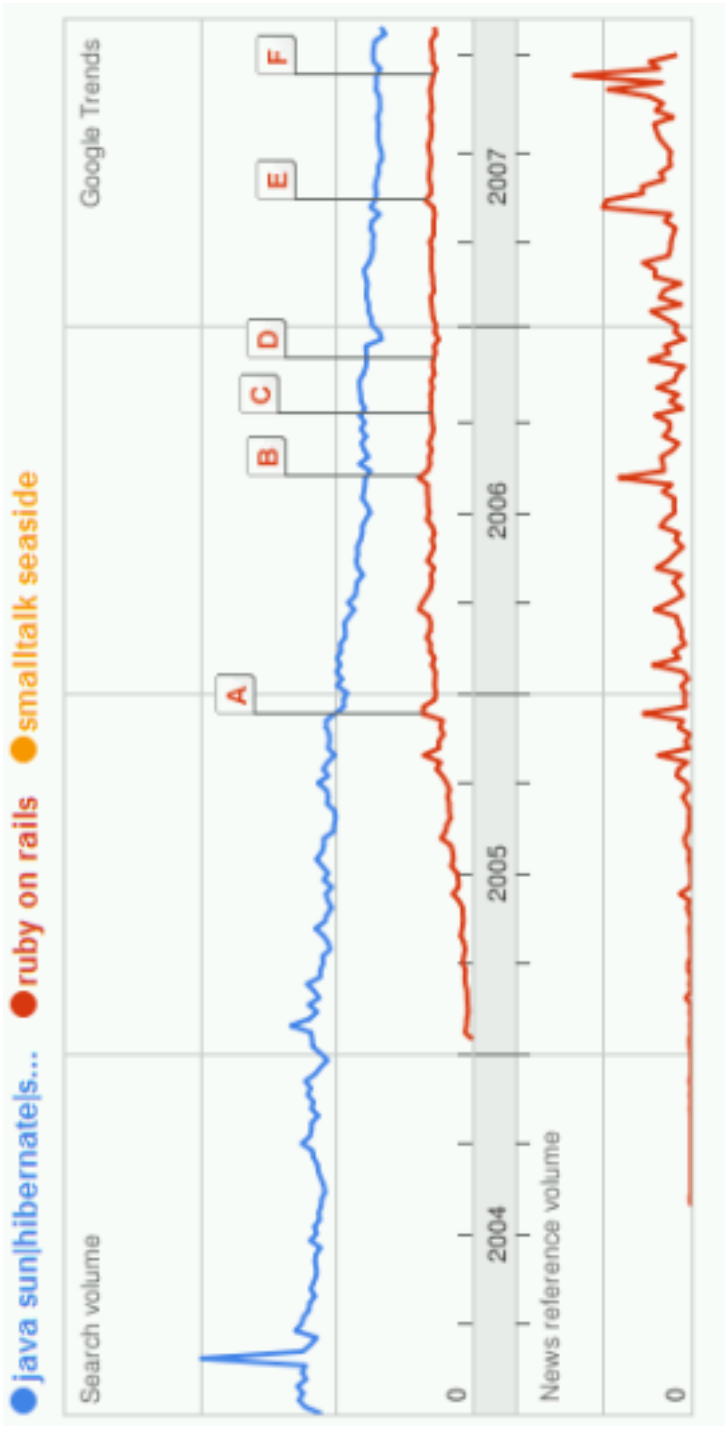

Abbildung 6.2.: Google Trends für Java, Rails, Seaside

Abbildungsverzeichnis

2.1. GISTA: Fachliche Architektur 8

5.1. Datenmodell-Entwurf des GISTA 54
5.2. Ruby on Rails Architektur 56
5.3. Geschäftsprozeß für einen Bestellvorgang in einem Internetshop 61

6.1. TIOBE Index 67
6.2. Google Trends für Java, Rails, Seaside 68

Tabellenverzeichnis

2.1. Anwendungsfälle des GISTA 7
2.2. Komponenten des GISTA 7
2.3. Verwendete Entwicklungsumgebung 9

3.1. Überblick Sprachkonzepte 22
3.2. Sprachen-Ähnlichkeitsprofil 22
3.3. Sprach-Konzepte und ihre Eignung als Untersuchungskonzepte 34

5.1. Aufgaben von Webframeworks 43
5.2. Auszug aus der JavaEE API 47
5.3. Spring Sub-Frameworks 48
5.4. Einige Rails-Projekt-Unterverzeichnisse 51
5.5. Ruby on Rails Sub-Frameworks 53
5.6. Produktivitätsvergleich Java/Spring/Hibernate und Ruby on Rails 59
5.7. Smalltalk Komponenten im Vergleich zu Ruby on Rails 60

Quellcodes

3.1. In Ruby ist alles ein Objekt . . . 13
3.2. Ruby ist stark, dynamisch typisiert . . . 13
3.3. Ruby Bereiche . . . 14
3.4. Java hat keine Bereiche . . . 14
3.5. Ruby Files . . . 14
3.6. Ruby Aliasing . . . 15
3.7. Java: Typsicherheit in Collections vor Java 5.0 . . . 17
3.8. Java: Typsicherheit in Collections nach Java 5.0 mit Generics . . . 17
3.9. Java Reflection . . . 18
3.10. Ruby Reflection . . . 19
3.11. Kontrollstrukturen in Smalltalk . . . 28

4.1. Ruby Mehrfachvererbung . . . 36
4.2. Java Reguläre Ausdrücke . . . 36
4.3. Ruby Iteration . . . 37
4.4. Java Iteration . . . 37
4.5. Smalltalk Iteration . . . 37
4.6. Ruby Codeanpassung . . . 38
4.7. Java Codeanpassung . . . 38
4.8. Smalltalk Codeanpassung . . . 39
4.9. Ruby Variante Routenziel . . . 39
4.10. Java Variante Routenziel . . . 39
4.11. Smalltalk Variante Routenziel . . . 39
4.12. Ruby Reflection . . . 40
4.13. Java Reflection . . . 40
4.14. Smalltalk Reflection . . . 41

5.1. Tracing in Java ohne AOP . . . 44
5.2. Tracing in Java mit AOP Aspekt . . . 44
5.3. Ruby Options Hash . . . 51
5.4. Ruby YAML DB Config File . . . 51
5.5. generierte Ruby Klasse . . . 52
5.6. Active Record Assoziationen . . . 53

5.7. Active Record Compositions 55
5.8. Active Record Inheritance 55
5.9. Geschäftsprozeß in Seaside 60

Glossar

Code Compile Cycle	Der Weg und die Schnelligkeit, mit der bei der Softwareentwicklung im Code gemachte Änderungen überprüft werden können.
Common Gateway Interface	Ein Standardprotokoll, auf dessen Grundlage der Aufruf externer Anwendungen von einem Webserver möglich ist. Dadurch kann dieser HTTP-Anfragen eines Browsers an jene externen Anwendungen umleiten und deren Output wieder an den Browser zurückschicken.
Domänenspezifische Programmiersprache	Eine domänenspezifische Programmiersprache (engl. domain-specific language, DSL) ist eine formale Sprache, die speziell für ein bestimmtes Problemfeld (die Domäne) entworfen und implementiert wird und daher eine größere Ausdruckskraft besitzt.
Fixture	Fixtures sind Testdaten. Sie werden beim Ausführen von Tests in die Testdatenbank geladen.
Geocoding	Bezeichnet den Vorgang, bei dem Adressangaben von geografischen Orten auf ein Paar von Längen- und Breitengraden abgebildet werden.
Java Community Process	Ein Verfahren, das bei der Weiterentwicklung der Programmiersprache Java und ihrer Standardbibliothek angewendet wird. Der Organisationsablauf wurde selbst durch den JCP definiert. Mitwirkende sind unter anderem die Firmen Sun, IBM, Apple

Laufzeitumgebung	Eine Softwareschicht, welche sich zwischen Anwendungs- und Betriebssystem-Schicht befindet. Dadurch, daß die Basisdienste Lesen und Schreiben von Dateien, Daten über Netzwerke transportieren, Ein- und Ausgabegeräte steuern, Daten verwalten und Sortieren und Suchen gekapselt werden, wird Plattformunabhängigkeit erreicht.
Scaffolding	Scaffolding ist englisch und heißt „Gerüst“. Im Kontext der Webentwicklung ist damit das automatische Generieren von grundlegenden immer wieder notwendigen Codegerüsten gemeint, auf denen sich bei der weiteren Entwicklung gut aufbauen läßt. Prototyping wird so unterstützt

Abkürzungsverzeichnis

AOP	Aspektorientierte Programmierung
AR	Active Record
ASCII	American Standard Code for Information Interchange
BIS	Built-in Security
CGI	Common Gateway Interface
CoC	Convention over configuration
CRUD	Create, Delete, Update, Destroy
DBC	Design by contract
DI	Dependency Injection
EJB	Enterprise Java Beans
FHO	Funktionen höherer Ordnung
GIS	Geo Informations System
GISTA	Geo Informationssystem für Trekking Aktivitäten
IDE	Integrated Development Environment
IoC	Inversion of Control
JavaEE	Java Platform Enterprise Edition
JCP	Java Community Process
JSP	Java Server Pages
JVM	Java Virtuelle Maschine
KVM	Klassenvariablen- und Methoden
LC	Lexikalische Closure
LOC	Lines of Code
MVC	Model View Controller
ORM	Object Relational Mapping
POJO	Plain old java object
RA	Reguläre Ausdrücke
REST	Representational State Transfer
RSS	Really Simple Syndication
RTE	Runtime Environment, zu deutsch: Laufzeitumgebung
SOAP	ursprünglich für Simple Object Access Protocol
TE	Type Erasure
u.a.	unter anderem

VM ..... virtuelle Maschine
XML ..... Extensible Markup Language
ZA ..... Zeigerarithmetik

Literaturverzeichnis

[AB] AB, MySQL: *Mysql Server 5*. – URL http://dev.mysql.com/downloads/

[Aptana] APTANA: *Radrails*. – URL http://www.radrails.org/

[Berlin] BERLIN, Uni: *Aspektorientierte Programmierung für die Praxis betrieblicher Softwareentwicklung*. – URL http://www.topprax.de/

[Bonn] BONN, Uni: *Publications: Aspect Oriented Programming*. – URL http://www.cs.uni-bonn.de/~gk/gk/?publications/by_date

[Bryant 2005] BRYANT, Avi: *Abstracting*. 2005. – URL http://www.cincomsmalltalk.com/userblogs/avi/blogView?showComments=true\&printTitle=Abstracting\&entry=3291042623

[Darmstadt] DARMSTADT, Uni: *Aspect Oriented Programming*. – URL http://www.st.informatik.tu-darmstadt.de/static/pages/projects/AORTA/Steamloom.jsp

[Ducasse u. a. 2004] DUCASSE, S. ; LIENHARD, A. ; RENGGLI, L.: Seaside - a multiple control flow web application framework. In: *Proceedings of ESUG Research Track* (2004), S. 231–257

[Foundation] FOUNDATION, Eclipse: *Eclipse - an open development platform*. – URL http://www.eclipse.org/

[Fowler u. a. 2003] FOWLER, M. u. a.: Patterns of Enterprise Application Architecture. (2003)

[Fowler 2005] FOWLER, Martin: InversionOfControl. In: *martinfowler.com* (2005). – URL http://martinfowler.com/bliki/InversionOfControl.html. – [Stand: 10. November 2007]

[Google a] GOOGLE: *Google Map API Concepts*. – URL http://code.google.com/apis/maps/documentation/

[Google b] GOOGLE: *Google Maps API Reference*. – URL http://code.google.com/apis/maps/documentation/reference.html

[Grosenbach] GROSENBACH, Geoffrey: *The Complete Guide to Rails Plugins: Part I*. – URL http://nubyonrails.com/articles/2006/05/04/the-complete-guide-to-rails-plugins-part-i

[Johnson 2002] JOHNSON, R.: *Expert One-on-One J2EE Design & Development*. Wrox Press Ltd. Birmingham, UK, UK, 2002

[Lewis u. a. 2007] LEWIS, A. ; PURVIS, M. ; SAMBELLS, J. ; TURNER, C.: Beginning Google Maps Applications with Rails and Ajax: From Novice to Professional (Beginning: from Novice to Professional). (2007)

[Meyer 1997] MEYER, B.: *Object-oriented software construction*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1997

[Opensource a] OPENSOURCE: *Interactive Ruby Shell*. – URL http://home.vr-web.de/juergen.katins/ruby/buch/irb.html. – Online-Stand: 05.11.2007

[Opensource b] OPENSOURCE: *One-Click Ruby Installer for Windows*. – URL http://rubyinstaller.rubyforge.org/wiki/wiki.pl

[Opensource c] OPENSOURCE: *RDT - Ruby Development Tools*. – URL http://rubyeclipse.sourceforge.net/download.rdt.html

[Opensource d] OPENSOURCE: *Ruby Core Library*. – URL http://corelib.rubyonrails.org/

[Opensource e] OPENSOURCE: *Ruby on Rails API*. – URL http://api.rubyonrails.org/

[Opensource f] OPENSOURCE: *Ruby Standard Library Documentation*. – URL http://stdlib.rubyonrails.org/

[Opensource g] OPENSOURCE: *Seaside*. – URL http://seaside.st/

[Schempp 2006] SCHEMPP, Ruben: *Aktuelle Entwicklungstrends im Bereich von Programmiersprachen*, Hochschule für angewandte Wissenschaften in Hamburg, Bachelorarbeit, 2006

[Subramaniam 2005] SUBRAMANIAM, Venkat: Generics in Java - Part I-III. In: *www.agiledeveloper.com* (2005). – URL http://www.agiledeveloper.com/articles/GenericsInJavaPartI.pdf. – für die anderen Teile ersetze man die römischen Zahlen im pdf entsprechend durch II oder III

[Tate 2005] TATE, B.A.: *Beyond Java*. O'Reilly, 2005

[TIOBE] TIOBE: *TIOBE Programming Community Index for November 2007*. – URL http://www.tiobe.com/tpci.htm

[Voegele] VOEGELE, Jason: *Programming Language Comparison*. – URL http://www.jvoegele.com/software/langcomp.html

[Wikipedia 2007a] WIKIPEDIA: *Aspektorientierte Programmierung — Wikipedia, Die freie Enzyklopädie*. 2007. – URL http://de.wikipedia.org/w/index.php?title=Aspektorientierte_Programmierung\&oldid=38685345. – [Online; Stand 10. November 2007]

[Wikipedia 2007b] WIKIPEDIA: *Ausnahmebehandlung — Wikipedia, Die freie Enzyklopädie*. 2007. – URL http://de.wikipedia.org/w/index.php?title=Ausnahmebehandlung&oldid=37624311. – [Online; Stand 31. Oktober 2007]

[Wikipedia 2007c] WIKIPEDIA: *Überladen — Wikipedia, Die freie Enzyklopädie*. 2007. – URL http://de.wikipedia.org/w/index.php?title=%C3%9Cberladen&oldid=32461249. – [Online; Stand 31. Oktober 2007]

[Wikipedia 2007d] WIKIPEDIA: *Closure — Wikipedia, Die freie Enzyklopädie*. 2007. – URL http://de.wikipedia.org/w/index.php?title=Closure\&oldid=35556002. – [Online; Stand 18. November 2007]

[Wikipedia 2007e] WIKIPEDIA: *Closure (computer science) — Wikipedia, The Free Encyclopedia*. 2007. – URL http://en.wikipedia.org/w/index.php?title=Closure_%28computer_science%%29\&oldid=169368643. – [Online; accessed 8-November-2007]

[Wikipedia 2007f] WIKIPEDIA: *Comparison of web application frameworks — Wikipedia, The Free Encyclopedia*. 2007. – URL http://en.wikipedia.org/w/index.php?title=Comparison_of_web_application_frameworks\&oldid=170400348. – [Online; accessed 12-November-2007]

[Wikipedia 2007g] WIKIPEDIA: *Continuation — Wikipedia, Die freie Enzyklopädie*. 2007. – URL http://de.wikipedia.org/w/index.php?title=Continuation\&oldid=36911534. – [Online; Stand 8. November 2007]

[Wikipedia 2007h] WIKIPEDIA: *Dependency injection — Wikipedia, The Free Encyclopedia*. 2007. – URL http://en.wikipedia.org/w/index.php?title=Dependency_injection\&oldid=166478294. – [Online; accessed 10-November-2007]

[Wikipedia 2007i] WIKIPEDIA: *Design By Contract — Wikipedia, Die freie Enzyklopädie*. 2007. – URL http://de.wikipedia.org/w/index.php?title=Design_By_Contract\&oldid=35854986. – [Online; Stand 18. November 2007]

[Wikipedia 2007j] WIKIPEDIA: *Design by contract — Wikipedia, The Free Encyclopedia*. 2007. – URL http://en.wikipedia.org/w/index.php?title=Design_by_contract&oldid=167179908. – [Online; accessed 7-November-2007]

[Wikipedia 2007k] WIKIPEDIA: *Diamond-Problem — Wikipedia, Die freie Enzyklopädie*. 2007. – URL http://de.wikipedia.org/w/index.php?title=Diamond-Problem\&oldid=37275481. – [Online; Stand 18. November 2007]

[Wikipedia 2007l] WIKIPEDIA: *Enterprise JavaBean — Wikipedia, The Free Encyclopedia*. 2007. – URL http://en.wikipedia.org/w/index.php?title=Enterprise_JavaBean\&oldid=169572275. – [Online; accessed 10-November-2007]

[Wikipedia 2007m] WIKIPEDIA: *Generische Programmierung — Wikipedia, Die freie Enzyklopädie*. 2007. – URL http://de.wikipedia.org/w/index.php?title=Generische_Programmierung&oldid=37175054. – [Online; Stand 31. Oktober 2007]

[Wikipedia 2007n] WIKIPEDIA: *Generische Programmierung — Wikipedia, Die freie Enzyklopädie*. 2007. – URL http://de.wikipedia.org/w/index.php?title=Generische_Programmierung\&oldid=38700022. – [Online; Stand 19. November 2007]

[Wikipedia 2007o] WIKIPEDIA: *Java Platform, Enterprise Edition — Wikipedia, Die freie Enzyklopädie*. 2007. – URL http://de.wikipedia.org/w/index.php?title=Java_Platform%2C_Enterprise_Edition&oldid=38122753. – [Online; Stand 10. November 2007]

[Wikipedia 2007p] WIKIPEDIA: *Java Virtual Machine — Wikipedia, Die freie Enzyklopädie*. 2007. – URL http://de.wikipedia.org/w/index.php?title=Java_Virtual_Machine&oldid=37171446. – [Online; Stand 26. Oktober 2007]

[Wikipedia 2007q] WIKIPEDIA: *List of web application frameworks — Wikipedia, The Free Encyclopedia*. 2007. – URL http://en.wikipedia.org/w/index.php?title=List_of_web_application_frameworks\&oldid=170299951. – [Online; accessed 9-November-2007]

[Wikipedia 2007r] WIKIPEDIA: *Reflection (computer science) — Wikipedia, The Free Encyclopedia*. 2007. – URL http://en.wikipedia.org/w/index.php?title=Reflection_%28computer_science%29&oldid=170287251. – [Online; accessed 19-November-2007]

[Wikipedia 2007s] WIKIPEDIA: *Ruby on Rails — Wikipedia, Die freie Enzyklopädie*. 2007. – URL http://de.wikipedia.org/w/index.php?title=Ruby_on_Rails&oldid=38424685. – [Online; Stand 11. November 2007]

[Wikipedia 2007t] WIKIPEDIA: *Seaside (software) — Wikipedia, The Free Encyclopedia*. 2007. – URL http://en.wikipedia.org/w/index.php?title=Seaside_%28software%29\&oldid=168400889

[Wikipedia 2007u] WIKIPEDIA: *Smalltalk-80 (Programmiersprache) — Wikipedia, Die freie Enzyklopädie*. 2007. – URL http://de.wikipedia.org/w/index.php?title=Smalltalk-80_%28Programmiersprache%29\&oldid=37504145. – [Online; Stand 17. November 2007]

[Wikipedia 2007v] WIKIPEDIA: *Web application framework — Wikipedia, The Free Encyclopedia*. 2007. – URL http://en.wikipedia.org/w/index.php?title=Web_application_framework&oldid=166583272. – [Online; accessed 26-October-2007]

[von www.javaCore.de] WWW.JAVACORE.DE, Wiklet Java W. von: *Reflection - Die Java Reflection API*. – URL http://wiklet.javacore.de/index.php/Reflection_-_Die_Java_Reflection_API. – Online-Stand: 17. November 2007

A. Sourcecode

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 19. November 2007 Alexander Seidl